

Politechnika Warszawska

W Y D Z I A Ł M A T E M A T Y K I
I N A U K I N F O R M A C Y J N Y C H



Praca dyplomowa inżynierska

na kierunku Informatyka

Interfejs użytkownika do manualnego obrysu struktur
oraz wybranych anormalności w obrazach medycznych
z wykorzystaniem tabletu graficznego

Łukasz Garstecki

Numer albumu 276857

Tomasz Świerczewski

Numer albumu 276915

Promotor

dr inż. Magdalena Jasionowska

WARSZAWA 2019

.....
podpis promotora

.....
podpisy autorów

Streszczenie

Niniejszy dokument prezentuje pracę dyplomową inżynierską pod tytułem "Interfejs użytkownika do manualnego obrysu struktur oraz wybranych anormalności w obrazach medycznych z wykorzystaniem tabletu graficznego". Celem realizowanym przez autorów pracy było stworzenie narzędzia do manualnego i półautomatycznego obrysowywania struktur anatomicznych, zmian patologicznych na obrazach medycznych, wykorzystywanego zarówno przez lekarzy, jak i inżynierów biomedycznych. Zakres pracy obejmuje omówienie: standardu DICOM, przegląd aplikacji dostępnych na rynku, pozwalających na przeglądanie i wykonywanie obrysów na obrazach medycznych DICOM, architektury tworzonego systemu i eksperymentów przeprowadzonych w trakcie tworzenia zaproponowanego narzędzia.

Zastosowana architektura umożliwia jednoczesne korzystanie z narzędzia przez wielu użytkowników. Moduł obrysów manualnych został zaimplementowany głównie w warstwie ściśle związanej z interfejsem użytkownika. Moduł obrysów półautomatycznych bazuje na algorytmie opracowanym na potrzeby generowania obrysów, który wykorzystuje operator Canny'ego oraz algorytm wyszukiwania najkrótszych ścieżek w grafie. Dodatkowo dodany moduł statystyczny umożliwia wyznaczenie podstawowych miar statystycznych, opisujących obrysowane fragmenty obrazów.

Słowa kluczowe: interfejs użytkownika, tablet graficzny, obrazy medyczne, DICOM, obrys, statystyki danych obrazowych, system informatyczny, interfejs REST API, wykrywanie krawędzi, generowanie obrysów

Abstract

Do nowego przetłumaczenia - Tomek.

Keywords: user interface, graphics tablet, medical images, DICOM, contour, statistics of image data, IT system, REST API interface, edge detection, contour generation

Warszawa, dnia

Oświadczenie

Oświadczam, że moją część pracy inżynierskiej (zgodnie z podziałem zadań opisanym we wstępie) pod tytułem „Interfejs użytkownika do manualnego obrysu struktur oraz wybranych anormalności w obrazach medycznych z wykorzystaniem tabletu graficznego”, której promotorem jest dr inż. Magdalena Jasionowska, wykonałem samodzielnie, co poświadczam własnoręcznym podpisem.

.....

Spis treści

Wstęp	11
1 Wprowadzenie	13
1.1 Zagadnienia medyczne związane z aplikacją	13
1.2 Podział prac	13
2 Stan wiedzy	15
2.1 Przegląd istniejących rozwiązań	15
2.1.1 Przeglądarki DICOM	15
2.1.2 Serwer z bazą plików DICOM	18
2.1.3 Podsumowanie	19
2.2 Proponowane rozwiązanie	19
3 Opis autorskiego narzędzia informatycznego	21
3.1 Specyfikacja wymagań	21
3.1.1 Opis biznesowy	21
3.1.2 Wymagania funkcjonalne	22
3.1.3 Wymagania niefunkcjonalne	23
3.2 Architektura rozwiązania	25
3.2.1 Interfejs użytkownika	26
3.2.2 Serwer obrazów Orthanc	26
3.2.3 Serwer obrysów i obliczeń	26
3.3 Moduł obrysów manualnych	27
3.4 Moduł obrysów półautomatycznych	27
3.4.1 Przegląd literatury	27
3.4.2 Opracowany algorytm obrysu półautomatycznego	29
3.4.3 Wygenerowanie bitmapy	30
3.4.4 Wykrycie krawędzi na bitmapie	30
3.4.5 Stworzenie grafu z bitmapy	32

3.4.6	Zapewnienie spójności grafu	35
3.4.7	Wyszukanie najkrótszych ścieżek w grafie	36
3.4.8	Optymalizacja algorytmu obrysu półautomatycznego	39
3.5	Moduł obliczeń statystyk	41
3.6	Moduł anonimizacji danych	42
4	Przeprowadzone eksperymenty	44
4.1	Zbiór testowy	44
4.2	Analiza działania aplikacji	45
4.3	Wydajność algorytmu półautomatycznego	45
4.4	Analiza wyników i wnioski	51
5	Podsumowanie	54
Bibliografia		56
Wykaz symboli i skrótów		60
Spis rysunków		64
Spis zawartości załączonej płyty CD		66
Spis załączników		68

Wstęp

W niniejszym rozdziale przedstawiono cel pracy inżynierskiej wraz z motywacją, która kierowała autorami przy wyborze właśnie tego tematu.

Motywacja

W czasach gwałtownego rozwoju sztucznej inteligencji jest ona stosowana prawie do każdej dziedziny z szeroko pojętej nauki. Jedną z takich dziedzin jest medycyna. Próba odnajdywania we wczesnym stadium różnych zmian patologicznych pozwala na szybką terapię, a w konsekwencji niejednokrotnie na całkowite wylecznie. Dlatego ważne jest komputerowe wspomaganie podejmowania decyzji w diagnostyce obrazowej. Obecnie trwają prace nad automatycznymi sposobami wykrywania takich zmian na obrazach medycznych.

W celu interpretowania obrazów medycznych potrzebna jest wiedza anatomiczna, jak i radiologiczna. Z tego powodu potrzeba stworzyć narzędzie, które w wygodny sposób pozwoli nam wykonywać obrysów wybranych narządów i anormalności medycznych. Na podstawie wykonanych obrysów będzie można w przyszłości tworzyć zaawansowane narzędzia do automatycznej klasyfikacji lub rozwijać zaproponowane narzędzie.

Cel pracy

Celem niniejszej pracy inżynierskiej było stworzenie narzędzia informatycznego, umożliwiającego tworzenie obrysów struktur anatomicznych czy zmian patologicznych na obrazach medycznych. Użytkownik może otwierać dane obrazowe w formacie DICOM i dokonywać obrysów manualnych lub półautomatycznych wybranych struktur lub anormalności. W przypadku obrysów manualnych użytkownik może tworzyć kilka obrysów jednocześnie w celu zaznaczania anormalności w konkretnej tkance. Użytkownik może korzystać z tabletu graficznego w celu wygodniejszego rysowania obrysów.

Został stworzony algorytm półautomatyczny do tworzenia obrysów półautomatycznych. Jest

to metoda półautomatyczna, ponieważ jest wspomagana przez człowieka, który wybiera punkty na obrazie medycznym. Algorytm ma za zadanie interpolować punkty wybrane przez użytkownika. Do algorytmu półautomatycznego generowania obrysu został użyty operator Canny'ego, a następnie dane przetwarzanie są poprzez algorytmy grafowe. Została przeprowadzona walidacja, czy ta metoda sprawdza się w celu usprawnienia rysowania obrysów manualnych.

Z zaproponowanego przez autorów rozwiązania mogą korzystać lekarze w codziennej pracy, studenci medycyny w celu nauki wykrywania zmian patologicznych, czy też lekarze i inżynierowie biomedyczni w trakcie współpracy w badaniach naukowych.

Zaproponowane przez autorów rozwiązanie opiera się na aplikacji przeglądarkowej i serwera przechowującego obrysy. To rozwiązanie pozwala na użycie go sieciach lokalnych, gdzie użytkownik lub użytkownicy na dowolnej stacji roboczej mogą wykonywać obrysy, które zapisywane są na wspólnym dla wszystkich użytkowników serwerze.

1. Wprowadzenie

W tym rozdziale zostało przedstawione wprowadzenie dotyczące formatu obrazów medycznych DICOM i aspektu medycznego pracy. Ponadto opisano podział prac wykonanych w trakcie tworzenia narzędzia informatycznego.

1.1. Zagadnienia medyczne związane z aplikacją

DICOM (ang. Digital Imaging and Communications in Medicine) [17] to norma opracowana dla potrzeb wymiany i interpretacji danych medycznych, które są najczęściej obrazami medycznymi. W postaci pliku DICOM mogą być zapisywane obrazy tomografii komputerowej, obrazowania metodą rezonansu magnetycznego, cyfrowej angiografii subtrakcyjnej, zdjęcia rentgenowskie, mammografia i wiele innych badań. Do wad tego standardu należy różnorodność metod implementacji. Powoduje to czasem sytuacje, w których nie można wyświetlić pliku na sprzęcie innego producenta, ze względu na różnice w sposobie implementacji. W takich sytuacjach należy ręcznie modyfikować pliki DICOM, aby mogły być poprawnie obsłużone. Ponadto liczba tagów zawartych w pliku DICOM stanowi barierę w sprawnym korzystaniu z tak zapisanych wyników badań.

1.2. Podział prac

Ze względu na złożoność realizowanego narzędzia dokonano podziału zdań pomiędzy członków zespołu.

Łukasz Garstecki wykonał:

- interfejs użytkownika w aplikacji przeglądarkowej,
- obsługę wczytywania plików medycznych do aplikacji przeglądarkowej,
- moduł anonimizacji danych pacjenta i modyfikacji plików medycznych,
- narzędzie do rysowania obrysu manualnego,

- narzędzie do wybierania punktów obrysu półautomatycznego,
- narzędzie do wizualizacji stworzonych obrysów przez użytkownika.

Tomasz Świerczewski wykonał:

- serwer zapisujący i generujący obrys wraz z bazą danych,
- algorytm generujący obrys półautomatyczne,
- REST API, udostępniające obrys,
- moduł statystyk,
- testy aplikacji.

2. Stan wiedzy

W poniższym rozdziale zostały przedstawione istniejące na rynku aplikacje o funkcjonalnościach określonych w celu pracy. Opisano wymagania zarówno funkcjonalne jak i niefunkcjonalne jakie powinien spełniać. Autorski system do tworzenia obrysów na obrazach w formacie DICOM struktur anatomicznych czy zmian patologicznych powinien spełniać wymagania zarówno lekarzy jak i inżynierów medycznych, wykorzystujących to narzędzie w pracy badawczej i naukowej. Przedstawiono również proponowane rozwiązanie postawionych przed systemem wymagań.

2.1. Przegląd istniejących rozwiązań

Obrysy to krzywe zamknięte naniesione na obraz, w celu oznaczenia wewnętrznej części. Obrysy można tworzyć manualnie (użytkownik wybiera wszystkie punkty obrysu), lub półautomatycznie (użytkownik wybiera kilka punktów, a algorytm łączy je na podstawie analizy obrazu). Obie metody są niedoskonałe, gdyż często to użytkownik musi skorygować obrys, ze względu na różnorodność obrazów medycznych wynikającą z różnych systemów obrazowania, czy też różnorodnej tekstury tkanki obrazowanej.

2.1.1. Przeglądarki DICOM

Istnieje wiele rozwiązań umożliwiających wykonywanie obrysów przy użyciu przeglądarki. W tej pracy opisano szczegółowo dwie wybrane aplikacje.

Pierwsze z nich to aplikacja OHIF Viewer oparta o biblioteki Cornerstone.

Cornerstone Core [8] to biblioteka, która umożliwia przetwarzanie obrazów medycznych z wykorzystaniem elementu canvas z języka HTML5. Biblioteka udostępnia interfejs do wyświetlania obrazów medycznych pozwalający na zarządzanie wyświetleniem zdjęcia. Podstawowe funkcjonalności obsługiwane przez bibliotekę to:

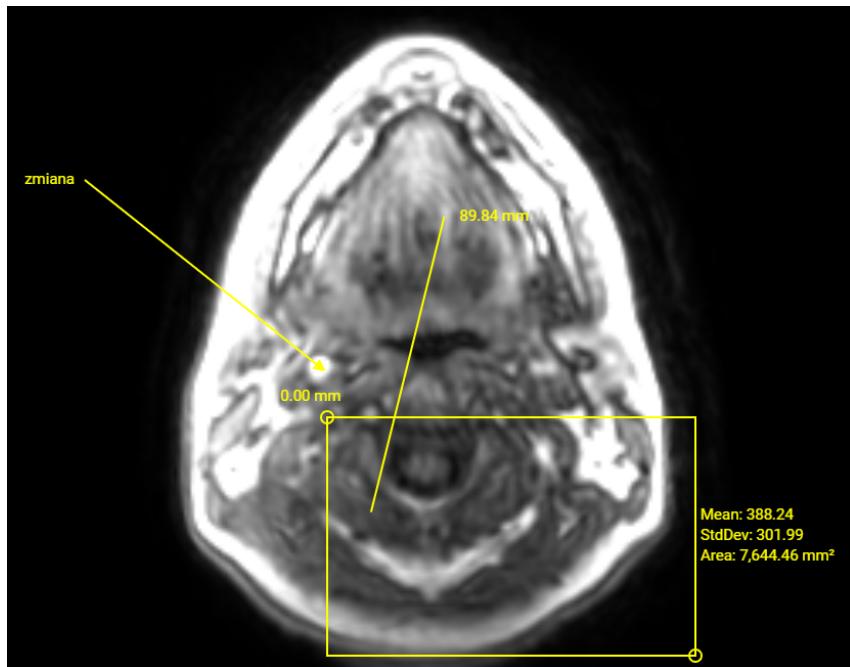
- przybliżanie i oddalanie obrazu,
- obrót obrazu,
- przesuwanie obrazu w wyświetlanym komponencie,

- zmiana jasności wyświetlanego obrazu,
- mapowanie kolorów,
- interpolacja pikseli w obrazie (dla obrazów o niskiej rozdzielczości).

Ponadto twórcy biblioteki Cornerstone Core stworzyli bibliotekę Cornerstone Tools, która korzysta z biblioteki Cornerstone Core i poza funkcjonalnościami Cornerstone Core udostępnia opisane niżej funkcjonalności potrzebne lekarzom do analizy badań pacjentów, takie jak:

- mierzenie odległości w linii prostej na obrazie z podaniem rzeczywistych wartości,
- oznaczanie obszarów przy pomocy prostokątów oraz elips,
- oznaczanie niewielkich zmian w postaci małego okręgu,
- mierzenie kątów na podstawie trzech podanych przez użytkownika punktów.

Przykładowym projektem korzystającym z bibliotek Cornerstone jest OHIF Viewer. Aplikacja pozwala na wykorzystanie większości możliwości udostępnianych przez biblioteki Cornerstone. Przykładowe użycie aplikacji zostało przedstawione na Rysunku 2.1.

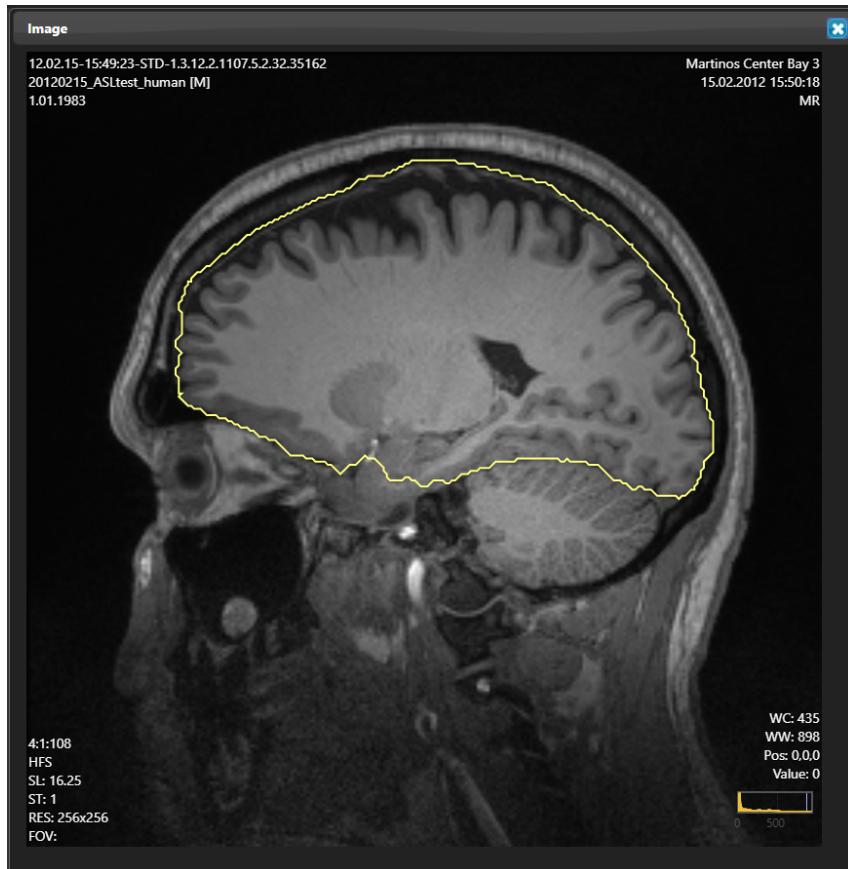


Rysunek 2.1: Przykład użycia aplikacji OHIF Viewer — oznaczono: prostokątny obszar, dla którego obliczone zostały wybrane statystyki; odcinek, dla którego otrzymano odległość jednostkach rzeczywistych; wskaźnik, wskazujący na zmianę, posiadający etykietę tekstową

Niestety aplikacja nie umożliwia wykonywania obrysów manualnych, ani półautomatycznych, jak również nie pozwala na zapisanie naniesionych na obraz oznaczeń.

Drugim rozwiązaniem działającym pozwalającym na pracę z plikami DICOM w przeglądarce internetowej to DICOM Web Viewer.

2.1. PRZEGŁĄD ISTNIEJĄCYCH ROZWIAZAŃ



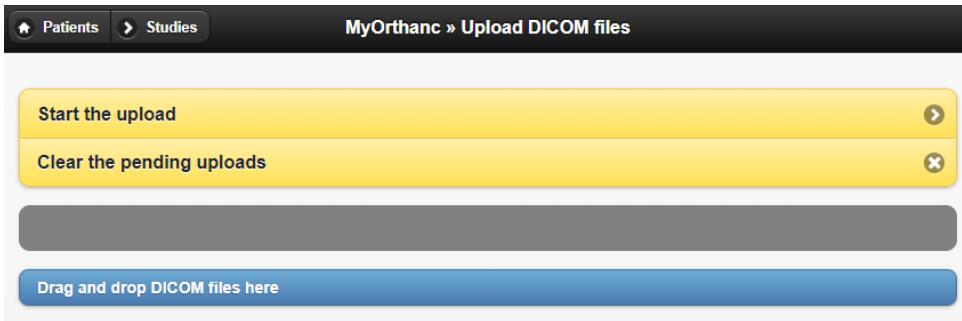
Rysunek 2.2: Przykład użycia aplikacji DWV - obrys półautomatyczny (Livewire)

Aplikacja DICOM Web Viewer (DWV) [10] jest również przykładem aplikacji przeglądarkowej służącej do przetwarzania obrazów DICOM, z wygodnym interfejsem automatycznego i półautomatycznego obrysowywania obszarów zaznaczonych przez użytkownika na obrazie. Przykładowy obrys wykonany przy użyciu aplikacji DWV przedstawiono na Rysunku 2.2.

Poza tym aplikacja udostępnia:

- przybliżanie i oddalanie obrazu,
- przesuwanie obrazu w wyświetlanym komponencie,
- zmiana jasności wyświetlanego obrazu,
- mierzenie odległości w linii prostej na obrazie z podaniem rzeczywistych wartości,
- oznaczanie obszarów przy pomocy prostokątów oraz elips.

Aplikacja nie pozwala na wykonywanie manualnych obrysów poprzez samodzielne poruszanie kursem po obrazie.



Rysunek 2.3: Interfejs wgrywania plików DICOM do serwera Orthanc

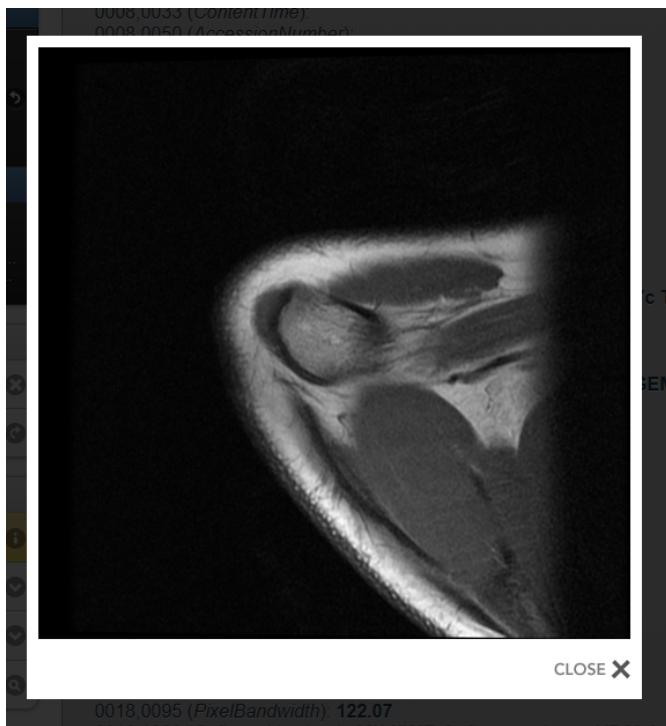
Rysunek 2.4: Podgląd tagów pliku DICOM przy użyciu interfejsu przeglądarkowego Orthanc

2.1.2. Serwer z bazą plików DICOM

Orthanc [19] to serwer plików DICOM, który umożliwia łatwe przechowywanie, zarządzanie oraz dostęp do plików medycznych DICOM. Ponadto Orthanc udostępnia REST API [20], które umożliwia przeglądanie wgranych na serwer plików DICOM podzielonych na kategorie takie jak pacjenci, badania i serie.

Serwer Orthanc udostępnia prosty interfejs webowy, dzięki któremu możliwe jest wgrywanie nowych plików (interfejs wgrywania plików przedstawiono na Rysunku 2.3), przeglądanie

2.2. PROPONOWANE ROZWIĄZANIE



Rysunek 2.5: Podgląd obrazu DICOM przy użyciu interfejsu przeglądarkowego Orthanc

zapisanych plików w szczególności tagów (Rysunek 2.4) oraz podglądzanie obrazów (Rysunek 2.5). Interfejs webowy nie zapewnia żadnej metody tworzenia na wybranym obrazie, ale pozwala na przełączanie pomiędzy kolejnymi obrazami za pomocą kliknięcia lewym przyciskiem myszy w prawą część podglądu obrazu.

2.1.3. Podsumowanie

Istniejące rozwiązania przeglądarkowe nie zapewniają wygodnego interfejsu użytkownikowi do manualnego obrysowywania interesujących go fragmentów obrazów medycznych przy użyciu tabletu graficznego. Ponadto żadne z narzędzi nie przechowuje utworzonych w sposób manualny ani wygenerowanych półautomatycznie obrysów.

2.2. Proponowane rozwiązanie

W rozwiążaniu zdecydowano się skorzystać z serwera Orthanc jako bazy przechowującej pliki DICOM. Przy decyzji miało znaczenie przede wszystkim to, że jest to narzędzie bezpłatne. Ponadto udostępniane przez API serwera funkcjonalności pozwalające na zarządzanie i przeglądanie zapisanych plików są wystarczające dla realizowanego systemu.

Poza serwerem Orthanc stworzono serwer w technologii .NET Core [12] wspomagający inter-

fejs użytkownika, który pozwala na przeprowadzanie czasochłonnych aplikacji poza interfejsem użytkownika. Wybrano tę technologię ze względu na to, że jest ciągle rozwijana, podobnie jak Orthanc .NET Core jest rozwiązaniem darmowym i nie ustępuje jakością w stosunku do Java EE [11]. Podstawowymi zadaniami tego serwera było przechowywanie wykonanych przez użytkownika obrysów manualnych, generowanie i przechowywanie obrysów półautomatycznych oraz obliczanie statystyk związanych z zapisanymi obrysami.

Jako interfejs użytkownika wykorzystano bibliotekę ReactJS [7] w połączeniu z biblioteką Redux[1]. Zdecydowano się na tę technologię ze względu na bardzo dobre wsparcie techniczne i elastyczność [18]. Wybranie tej technologii pozwoliło na stworzenie aplikacji przeglądarkowej pozwalającej na generowanie obrysów manualnych przy użyciu myszy lub tabletu graficznego, wybieranie punktów, z których generowane będą obrysy półautomatyczne, oglądanie wcześniej wygenerowanych i zapisanych obrysów oraz wyświetlanie statystyk dotyczących wykonanego obrysu. Interfejs inspirowany jest podglądem obrazu w aplikacji DWV, ale działanie półautomatycznego obrysu zostało zrealizowane inaczej — obrys nie jest generowany na bieżąco, lecz na życzenie użytkownika. Wybrane przez niego punkty na obrazie wysyłane są do serwera i na obrazie wyświetlany jest wynik półautomatycznego dopasowania obrysu.

3. Opis autorskiego narzędzia informatycznego

W poniższym rozdziale zawarto dokumentację techniczną i biznesową tworzonego systemu. Przedstawiono w szczególności: wymagania funkcjonalne i niefunkcjonalne, architekturę, zastosowane metody półautomatycznego obrysu oraz metody obliczania statystyk obrysu.

3.1. Specyfikacja wymagań

Przed stworzeniem systemu zostały zgromadzone dane dotyczące wymagań stawianych przed tworzonym systemem. Zostały one przedstawione w postaci zgodnej z wymaganiami stawianymi w inżynierii oprogramowania w rozdziałach 3.1.1. - 3.1.3. Wymagania były gromadzone na podstawie wywiadów przeprowadzonych z opiekunem naukowym tej pracy dyplomowej, z lekarzami i ze studentami Warszawskiego Uniwersytetu Medycznego.

3.1.1. Opis biznesowy

Celem projektu jest stworzenie interfejsu przyjaznego użytkownikowi, który umożliwi przeglądanie plików DICOM, a także przeprowadzanie na tych plikach obrysów. Prace obejmują stworzenie aplikacji webowej, która udostępnii użytkownikowi interfejs komunikujący się z bazą danych Orthanc. Ponadto prace obejmują stworzenie serwera odpowiedzialnego za przechowywanie wygenerowanych przez użytkownika obrysów oraz wyznaczanie obrysów półautomatycznych.

Do podstawowych funkcjonalności systemu zaliczają się:

- generowanie obrysu manualnego.
- generowanie obrysu półautomatycznego na podstawie punktów wybranych przez użytkownika.
- zapisywanie wygenerowanych obrysów.
- anonimizacja¹ danych zapisanych w strukturze pliku DICOM.

¹Anonimizacja (ang. anonymization) — operacja mająca na celu usunięcie z danych informacji o pacjentach, które pozwoliłyby na identyfikację danych z tożsamością pacjenta. Są to między innymi: imiona, nazwisko, pesel. Inne tłumaczenia słowa anonymization — utajnianie, usuwanie danych niejawnych. Z uwagi na fakt, że te tłumaczenia nie oddają dobrze kontekstu, zastosowano kalkę językową.

3. OPIS AUTORSKIEGO NARZĘDZIA INFORMATYCZNEGO

3.1.2. Wymagania funkcjonalne

Nieodłącznym elementem inżynierii oprogramowania przy próbie dokumentacji określonego produktu lub usługi, są wymagania funkcjonalne i niefunkcjonalne. Te pierwsze opisują funkcje i możliwości, które system powinien realizować. Zostały przygotowane wymagania funkcjonalne dla tworzonego systemu. Przedstawiono je poniżej w postaci historii użytkownika (ang. user stories), czyli czynności jakie może chcieć wykonać użytkownik tego systemu.

1. Jako użytkownik chcę wczytać obraz DICOM.

Użytkownik może wybrać obraz w menu bocznym, w którym ma możliwość wyboru pacjenta, badania oraz serii. Wybranie serii skutkuje wyświetleniem pierwszego obrazu DICOM z tej serii.

2. Jako użytkownik chcę zmienić obraz w serii przy użyciu rolki myszy.

Po umieszczeniu kurSORA na obrazie, przewijanie rolką myszy do góry powoduje zmianę wyświetlanego obrazu na kolejny obraz w serii. Gdy przewijamy rolką myszy do góry na ostatnim obrazie w serii wyświetlany obraz nie zmienia się. Analogicznie przewijanie rolką myszy w dół powoduje zmianę wyświetlanego obrazu na poprzedni obraz w serii, a przewijanie w dół rolką myszy na pierwszym obrazie w serii nie powoduje zmiany obrazu.

3. Jako użytkownik chcę wykonać obrys przy użyciu tabletu graficznego.

Po umieszczeniu kurSORA na obrazie sterowanym przez tablet graficzny, użytkownik prowadzi kurSOR po obrazie wykonując obrys bez odrywania końcówki rysika od podkładki. Jeżeli użytkownik nie zakończy obrysu dokładnie w punkcie, w którym go rozpoczął, obrys powinien zakończyć się linią prostą, łączącą punkt końcowy z punktem początkowym.

4. Jako użytkownik chcę wygenerować obrys na podstawie wybranych punktów.

Po umieszczeniu kurSORA na obrazie użytkownik może wybierać punkty, na podstawie których zostanie wygenerowany obrys, poprzez wcisnięcie lewego przycisku myszy w miejscach, w których chce, aby znalazły się punkty. Użytkownik może zobaczyć efekt wygenerowanego przez system obrysu.

5. Jako użytkownik chcę edytować listę punktów, z której wygenerowany zostanie obrys.

Użytkownik może usunąć wcześniej wybrany punkt po umieszczeniu nad nim kurSORA i wcisnięciu lewego przycisku myszy. Użytkownik może dodać nowy punkt do listy punktów poprzez wcisnięcie lewego przycisku myszy w miejscu, w którym chce wstawić punkt.

3.1. SPECYFIKACJA WYMAGAŃ

6. Jako użytkownik chcę wybrać kolor obrysu.

Użytkownik wybiera kolor z palety kolorów lub może zdefiniować własny kolor poprzez podanie kodu RGB koloru, który chce wybrać.

7. Jako użytkownik chcę zapisać obrys.

Po wykonaniu obrysu manualnego lub wybraniu listy punktów do wygenerowania obrysu półautomatycznego, użytkownik wybiera nazwę obrysu i zapisuje obrys w systemie.

8. Jako użytkownik chcę obejrzeć zapisany obrys.

Użytkownik wybiera z listy obrysów zapisany obrys i przegląda obrys naniesiony na obraz, na którym został wykonany.

9. Jako użytkownik chcę zobaczyć statystyki dotyczące obrysu.

Użytkownik wybiera z listy obrysów zapisany obrys i przegląda statystyki obliczone na podstawie zapisanego obrysu. Do statystyk zalicza się obwód obrysu, pole obrysu, histogram obrazu na obszarze obrysu oraz liczba pikseli wewnętrz obrysu.

10. Jako użytkownik chcę zobaczyć jednocześnie dowolną liczbę zapisanych w systemie obrysów na jednym obrazie DICOM.

Użytkownik wybiera poprzez kliknięcie lewym przyciskiem myszy na nazwie obrysu znajdującej się na liście obrysów. Wybrane obrysy wyświetlane są jednocześnie na przeglądany przez użytkownika zdjęciu. Użytkownik może wyłączyć podgląd wcześniej wybranego obrysu poprzez ponowne wcisnięcie lewego przycisku myszy na nazwie obrysu na liście po prawej stronie. Na zdjęciu wyświetlane są jedynie obrysy wykonane na tym obrazie.

11. Jako użytkownik chcę zanomizować dane pacjenta zawarte w pliku DICOM.

Użytkownik może zanomizować pacjenta, gdy przegląda jego obraz. Użytkownik może anonimizować imię i nazwisko pacjenta, datę urodzenia pacjenta oraz płeć pacjenta poprzez nadanie nowych wartości lub poprzez usunięcie poprzedniej wartości i pozostawienie pustych pól w formularzu.

3.1.3. Wymagania niefunkcjonalne

Drugim rodzajem wymagań są wymagania niefunkcjonalne. Opisują one kryteria osądzenia działania systemu pod kątem jakościowym. Założone wymagania niefunkcjonalne zostały przedstawione w Tabeli 3.1.

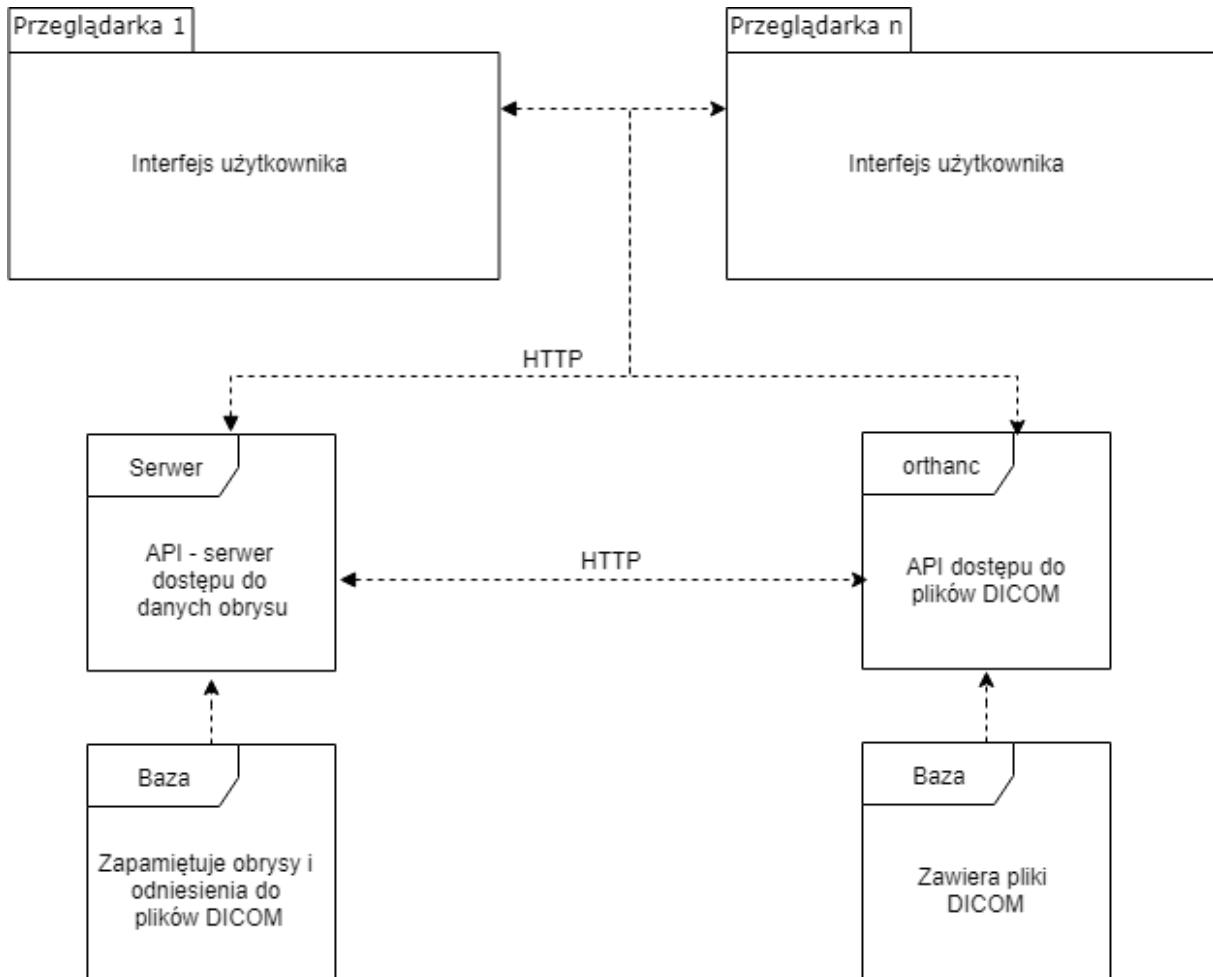
3. OPIS AUTORSKIEGO NARZĘDZIA INFORMATYCZNEGO

Tabela 3.1: Spis wymagań niefunkcjonalnych

Obszar wymagań	Opis
Użyteczność (ang. Usability)	Każda funkcjonalność aplikacji dostępna dla użytkownika musi mieścić się na pojedynczym ekranie przy rozdzielcości 1920x1080 i czcionce nie mniejszej niż 12pt.
	Aplikacja powinna udostępniać pobranie zapisanych obrysów przy użyciu serwisu REST.
Niezawodność (ang. Reliability)	Aplikacja ma być dostępna 24h w ciągu doby. Dopuszczalny jest brak działania aplikacji w dowolnym momencie przez okres nie dłuższy niż przez 12h. Po przerwie w działaniu aplikacja musi być dostępna przez kolejne 24h bez utrudnień.
Wydajność (ang. Performance)	Aplikacja powinna pobierać dane zewnętrzne w postaci pliku DICOM (około 20MB) nie dłużej niż 5 sekund.
	Aplikacja powinna generować obrys półautomatyczny i zapisywać obrys do systemu w czasie nie dłuższym niż 30 sekund.
	Aplikacja powinna reagować na działanie użytkownika (z wyjątkiem generowania obrysów półautomatycznego i zapisu obrysów do systemu) w czasie nie dłuższym niż 1 sekunda.

3.2. Architektura rozwiązania

Na Rysunku 3.1 przedstawiono architekturę autorskiego systemu.



Rysunek 3.1: Diagram UML przedstawiający architekturę autorskiego systemu

Architektura pozwala na połączenie z serwerem plików DICOM (Orthanc) oraz serwerem obrysów dowolnej liczby klientów korzystających z przeglądarki. Algorytmy zapisu obrysów, generowania obrysów, obliczania statystyk oraz anonimizacji plików są od siebie niezależne i mogą zostać zrównoleglane — ograniczeniem jest tutaj moc obliczeniowa maszyny na której uruchomiony jest serwer. Niemożliwe jest zwiększenie wydajności poprzez zwiększenie liczby instancji serwera. Dopuszczalne jest takie rozwiązanie, gdyż planowana liczba użytkowników jednocześnie korzystających z aplikacji to nie więcej niż 20 osób. Możliwości zwiększenia skalowalności w kontekście obliczeniowej zostało opisane w rozdziale 5.2.

3.2.1. Interfejs użytkownika

Kluczowym elementem stworzonego systemu jest interfejs użytkownika napisany w języku skryptowym JavaScript przy użyciu biblioteki ReactJS. Udostępnia ona interfejs nawigujący po zapisanych w serwerze Orthanc plikach DICOM. Ponadto dla wyświetlnego obrazu udostępnia szczegóły dotyczące obrazu, badania i pacjenta. Umożliwia także wykonanie obrysu manualnego zrealizowanego przy użyciu biblioteki react-canvas-draw. W aplikacji zawarto również autorski interfejs do wyboru punktów do algorytmu półautomatycznego, wykonany z użyciem elementu canvas [16]. Podgląd zapisanych w systemie obrysów wraz z wyliczonymi dla nich statystykami jest możliwy z poziomu tworzenia obrysów oraz w osobnej zakładce, gdzie udostępniono widok, w którym użytkownik może oglądać jednocześnie wybrane przez siebie obrysy z listy obrysów na jednym obrazie.

3.2.2. Serwer obrazów Orthanc

Wyświetlane w interfejsie użytkownika obrazy DICOM oraz związane z nimi informacje przechowywane są przez serwer DICOM. Dostęp do zapisanych na serwerze danych jest możliwy poprzez udostępniane przez serwer API. Interfejs użytkownika wykorzystuje API poprzez protokoł HTTP.

API serwera udostępnia dane obrazów posegregowanych względem kolejno: pacjentów, badań oraz serii. Wykorzystano je w interfejsie użytkownika do nawigacji pomiędzy obrazami. Dodatkowo dla każdego obrazu serwer udostępnia dane obrazowe, umożliwiając wyświetlenie obrazu w interfejsie użytkownika. Ponadto API udostępnia wszystkie tagi obrazu DICOM, co umożliwia wyświetlanie interfejsowi użytkownika informacji w nich zawartych.

3.2.3. Serwer obrysów i obliczeń

Kolejny kluczowy element stworzonego systemu to serwer obrysów i obliczeń. Udostępnia on API z obrysami, które zostało wykorzystane przez interfejs użytkownika. API zostało stworzone przy pomocy ASP.NET Core 2.2 [15] przy wykorzystaniu framework'a .NET Core 2.2 [14]. API oferuje 4 podstawowe operacje CRUD - utwórz, odczytaj, aktualizuj i usuń. W celu pobrania zdjęć medycznych serwer z API łączy się do bazy danych Orthanc. Informacje o obrysach, takie jak ID obrysu, tag, ID obrazu medycznego i typ obrysu (półautomatyczny lub manualny) są przechowywane w bazie danych Microsoft SQL Server. Pozostałe informacje, takie jak lista pikseli należących do obrysu, lista punktów w przypadku obrysu półautomatycznego i statystyki są przechowywane w plikach CSV. Te pliki są zapisywane w katalogu roboczym obok miejsca przechowywania plików źródłowych dla API.

3.3. MODUŁ OBRYSÓW MANUALNYCH

Poza serwerem Orthanc interfejs korzysta także z serwera przechowującego obrysów, a także wykonującego obliczenia związane z generowaniem obrysów. Serwer ten jest dostępny dla interfejsu użytkownika poprzez API dostępne przez protokół HTTP. Pozwala ono na pobieranie zapisanych obrysów, zapisywanie obrysów manualnych i generowanie podglądu i zapis obrysu półautomatycznego.

3.3. Moduł obrysów manualnych

Obrysów manualne generowane przez użytkowników zapisywane są przez serwer obrysów i obliczeń w postaci plików CSV na dysku serwera. W momencie zapisu obliczane są również statystyki dotyczące obrysowanej części obrazu.

W związku z tym, że obrazy wyświetlane w interfejsie są w innej rozdzielczości niż rzeczywisty rozmiar obrazu, obrys przed zapisaniem zostają przeskalowane do faktycznych wymiarów obrazu. W przypadku obrazów o małych rozdzielczościach może spowodować to utratę dokładności obrysu, a w przypadku obrysów bardzo małych obszarów zdegradowanie obrysu do jednego piksela. Tego typu straty mogą wpływać na poprawność obliczanych statystyk jak również błędne wyświetlanie zapisanych obrysów w interfejsie użytkownika.

3.4. Moduł obrysów półautomatycznych

Moduł obrysów półautomatycznych pozwala użytkownikom na generowanie obrysów poprzez wybieranie punktów na obrazie DICOM. Wybrane punkty łączone są ze sobą na podstawie autorskiego algorytmu opisanego w tym podrozdziale.

3.4.1. Przegląd literatury

Często na obrazach medycznych różnice w charakterystyce poziomów szarości pikseli reprezentujących interesujące nas obiekty są nieznaczne. Pliki te często nie posiadają dodatkowych informacji o położeniu tych obiektów, ani czym się wyróżniają i co przedstawiają. Problem opracowania uniwersalnego algorytmu wykrywania krawędzi na obrazach medycznych jest problemem trudnym. Takie algorytmy muszą spełniać szereg wymagań, które stwierdzają poprawność danego algorytmu, jak również operatora morfologicznego. Dla wielu takich algorytmów można znaleźć odpowiednie przykłady, dla których krawędzie nie zostaną wyznaczone poprawnie. Zgodnie z J. Cytowski [6] dobry detektor krawędzi powinien spełniać następujące warunki:

3. OPIS AUTORSKIEGO NARZĘDZIA INFORMATYCZNEGO

- niskie prawdopodobieństwo zaznaczenia punktów nienależących do krawędzi oraz niskie prawdopodobieństwo niezaznaczenia punktów należących do krawędzi,
- zaznaczone punkty krawędzi powinny być możliwie blisko jej osi,
- wyłącznie jedna odpowiedź na pojedynczy punkt krawędzi.

Detektory krawędzi oparte na gradiencie, w tym operator Canny'ego są często używane. Ich główne zalety na podstawie [6] to:

- dają dobre wyniki dla obrazów o dobrej jakości i bez szumów.
- są wydajne - ich złożoność jest liniowa względem liczby przetwarzanych pikseli.
- nie wymagają skomplikowanej sztucznej inteligencji do działania.

Zgodnie z [6] za ich główne wady można uznać:

- potrzebę określenia rozmiaru maski i wartości progowej, gdyż rozmiar maski znacząco wpływa na rozmieszczenie pozycji, w których gradient osiąga wartości maksymalne lub przecina zera,
- pomijanie narożników, które spowodowane jest faktem, że wartość 1D gradientu w narożnikach jest zazwyczaj zbyt mała, aby wykrywać wokół nich,
- znajdowanie schodkowych krawędzi, które są wykrywane tylko przez operator pierwszej pochodnej,
- dużą wrażliwość na szum,
- na podstawie obserwacji działania algorytmu — rozmyte krawędzie często nie są wykrywane przez małe różnice w wartościach kolejnych sąsiadujących pikseli.

Operator Canny'ego [5] składa się z 3 zasadniczych kroków:

1. Określenie wartości i kąta gradientu

W tym celu został wykorzystany operator gradientu, a estymatorem gradientu w funkcji dyskretniej, jaką jest obraz, zastosowano maskę, czy też operator Sobela. Wykorzystując go uzyskano dla każdego piksela wielkość oraz kierunek gradientu, co służy do dalszych obliczeń,

2. Wykrycie miejsc występowania krawędzi

W tym celu został wykorzystany algorytm usuwania niemaksymalnych pikseli (ang. *non-max suppression*). Polega on na wyborze takich pikseli, które mają największą wartość gradientu na linii o kierunku zgodnym z kątem danego gradientu. Możliwe są cztery kierunki: pionowy, poziomy oraz dwa diagonalne. Jeśli dany piksel miał większą wartość gradientu od dwóch swoich sąsiadów, to zaznaczono go jako potencjalny punkt tworzący krawędzie.

W ten sposób otrzymano obraz z potencjalnymi krawędziami,

3. Wyznaczanie krawędzi progowaniem histerezy

Po poprzednim kroku na obrazie nadal znajdują się nieistotne krawędzie. W tym celu Canny wprowadził ideę progowania histerezy. Metoda ta wymaga dwóch wartości progowych T_1, T_2 takich, że $T_1 < T_2$. Jeżeli wartość gradientu w danym pikselu jest większa od T_2 , to zaznaczono ten punkt jako krawędź. Jeśli tak się stało, to zaczęto proces śledzenia krawędzi — dla każdego sąsiada, którego wartość gradientu jest większa od T_1 zaznaczono go jako krawędź. Jest ona wykonywana rekurencyjnie dla każdego zakwalifikowanego punktu.

Zamiast dokładnych wartości progowych można przekazać do funkcji dwie wartości — t_1, t_2 , które są procentem liczby pikseli, które będą niedopuszczone jako krawędzie. Dla $t_1 = 0.7, t_2 = 0.9$ dopuszczono tylko 10% pikseli jako te, które są większe od T_2 . Podając t_1, t_2 wyznaczono rozkład wartości gradientu w badanym obrazie, obliczono dystrybuantę $F(x)$ i wybrano dla T_1 ten argument, dla którego $F(x) = t_1 * p$, gdzie p to liczba pikseli i analogicznie dla T_2 . W ten sposób wyznaczono progi do histerezy.

Operator Sobela [25] to metoda wyznaczania gradientu, a więc zarazem krawędzi w kierunku poziomym, jak i w pionowym. Dla każdego piksela przeprowadzono operację morfologiczną z następującymi maskami:

Maska rzędów	Maska kolumn
−1 −2 −1	−1 0 1
0 0 0	−2 0 2
1 2 1	−1 0 1

Po wykonaniu tych operacji otrzymano wartości s_1 i s_2 odpowiednio dla maski rzędów i kolumn. Na podstawie tych danych otrzymano następujące informacje o gradiencie:

Wielkość gradientu	Kierunek krawędzi
$\sqrt{s_1^2 + s_2^2}$	$\operatorname{tg}^{-1} \left[\frac{s_1}{s_2} \right]$

3.4.2. Opracowany algorytm obrysu półautomatycznego

Opracowany algorytm służy do półautomatycznego tworzenia obrysów na podstawie wybranych punktów, które są interpolowane przez algorytm bazujący na operatorze Cany'ego, operatorze Sobela i algorytmie A*. Powyższe rozwiązania pozwalają na uzyskanie najlepszych efektów na przetwarzanych przez narzędzie obrazach medycznych.

Danymi wejściowymi są:

3. OPIS AUTORSKIEGO NARZĘDZIA INFORMATYCZNEGO

- identyfikator obrazu medycznego, na którym był wykonywany obrys,
- lista punktów wybranych przez użytkownika; punkty te zostały wcześniej przeskalowane ze współrzędnych w internetowej aplikacji webowej na współrzędne odpowiadające rozdzielczości oryginalnego obrazu.

Algorytm można podzielić na kilka ważnych etapów:

- wygenerowanie bitmapy
- wykrycie krawędzi na bitmapie,
- stworzenie grafu z bitmapy,
- zapewnienie spójności grafu,
- wyszukanie najkrótszych ścieżek w grafie.

Poniżej zostaną przedstawione dokładne rozwiązania dla każdego z tych kroków.

3.4.3. Wygenerowanie bitmapy

Przed rozpoczęciem przetwarzania jest pobierany obraz medyczny o danym wcześniej identyfikatorze z serwera Orthanc. Bitmapa tworzona jest na podstawie obrazu DICOM poprzez zapisanie w odpowiednich komórkach wartości kolorów z oryginalnego obrazu. Stawi ona podstawę do dalszej pracy algorytmu.

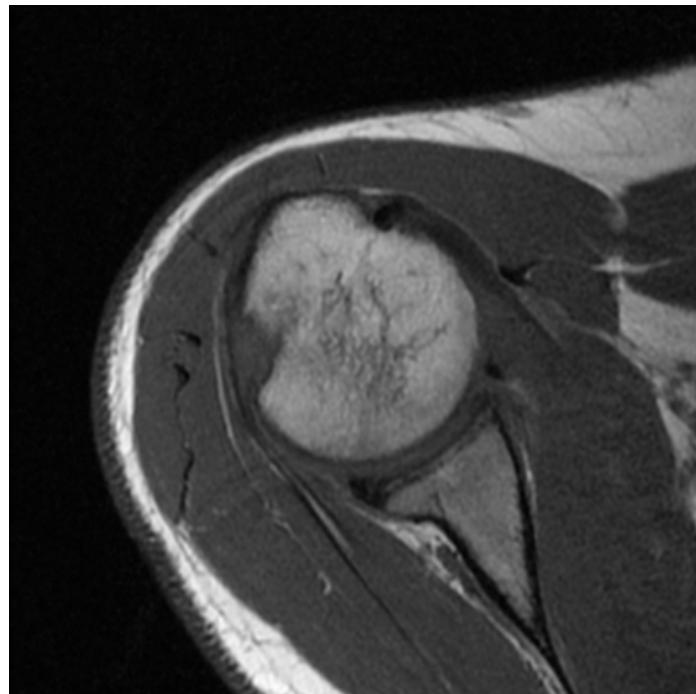
3.4.4. Wykrycie krawędzi na bitmapie

W pracy został użyty operator Canny'ego, ze względu na możliwość dostosowania go do bardzo różnorodnych problemów. Zgodnie z Rysunkiem 4.25 „Porównanie operatorów wykrywania krawędzi” w [6] najlepiej wykrywał główne narządy, takie jak wątroba czy też trzustka. Z wyżej wymienionych powodów został on wykorzystany w tym algorytmie.

Stosując operator Canny'ego otrzymano macierz, gdzie każde pole w macierzy odpowiada pikselowi w wejściowej bitmapie — obrazie medycznym. Jeśli w komórce macierzy znajduje się 1, to w tym miejscu na bitmapie znajduje się krawędź, w przeciwnym przypadku 0. W ten sposób algorytm wykrył wszystkie znaczące krawędzie na bitmapie. Kolejnym krokiem przetwarzania było stworzenie grafu na podstawie wyżej wymienionej macierzy.

Przykład działania operatora Canny'ego przedstawiono na Rysunkach od 3.2 do 3.4. Obraz wyjściowy do dalszych działań został przedstawiony na Rysunku 3.2.

3.4. MODUŁ OBRYSÓW PÓŁAUTOMATYCZNYCH



Rysunek 3.2: Oryginalny obraz, który przedstawia badanie rezonansem magnetycznym stawu ramiennego

Po dwóch pierwszych krokach operatora Canny'ego, czyli wyznaczeniu gradientu i zastosowaniu algorytmu usuwania niemaksymalnych krawędzi uzyskano obraz z potencjalnymi krawędziami. Został on przedstawiony na Rysunku 3.3.



Rysunek 3.3: Obraz po zastosowaniu algorytmu usuwania niemaksymalnych krawędzi

3. OPIS AUTORSKIEGO NARZĘDZIA INFORMATYCZNEGO

W ostatnim kroku była przeprowadzana histereza. Wynikiem jej działania jest macierz z informacjami gdzie znajduje się krawędź. Na Rysunku 3.4 wartościom 1 w macierzy odpowiada kolor czarny, a wartościom 0 odpowiada kolor biały.



Rysunek 3.4: Wynikowa macierz po zastosowaniu operatora Canny'ego, gdzie kolorem białym są reprezentowane wartości odpowiadające 0, a kolor czarny odpowiada 1

3.4.5. Stworzenie grafu z bitmapy

Kolejnym etapem było stworzenie grafu z bitmapy. Na tym etapie danymi wejściowymi były:

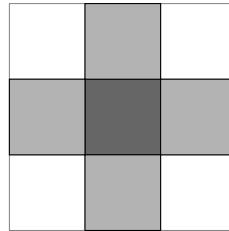
- macierz z wartościami logicznymi prawda/fałsz określającymi czy znajduje się w danym punkcie krawędź,
- punkty wybrane przez użytkownika aplikacji.

Przed rozpoczęciem działania algorytmu należy zapewnić łączność 4-krotną (ang. *pixel 4-connectivity*) [21], zwaną także sąsiedztwem von Neumanna. Przy łączności 4-krotnej sprawdza się tylko sąsiadów w poziomie lub pionie zgodnie ze schematem na Rysunku 3.5.

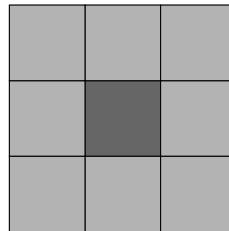
Dla łączności 8-krotnej sprawdza się wszystkich możliwych sąsiadów, także po przekątnej. Jest ona zwana także sąsiedztwem Moore'a lub otoczeniem Moore'a [26]. Została ona przedstawiona na Rysunku 3.6.

W przypadku zastosowania łączności 8-krotnej przy wyznaczaniu długości krawędzi trzeba zastosować metrykę Czebyszewa, która jest specjalnym przypadkiem odległości Minkowskiego.

3.4. MODUŁ OBRYSÓW PÓŁAUTOMATYCZNYCH



Rysunek 3.5: Ilustracja łączności 4-krotnej



Rysunek 3.6: Ilustracja łączności 8-krotnej

Przy zastosowaniu łączności 4-krotnej długość krawędzi jest obliczana zgodnie z metryką miejską, zwaną też metryką Manhattan.

Metryka Manhattan w kontekście dalszego przetwarzania w celu wyszukiwania najkrótszych ścieżek w grafie jest bardziej adekwatna, ponieważ jest intuicyjna w wyznaczaniu odległości na obrazie płaskim w porównaniu do metryki Czebyszewa. W tym przypadku najlepsza byłaby tutaj metryka Euklidesa, lecz mamy do czynienia nie z kolejnymi punktami oddalonymi od siebie, a z sąsiadującymi pikselami. Ponadto w tym algorytmie istotne jest szybkie szacowanie odległości, czy też długości danej krawędzi.

Wykrywanie wierzchołków przy łączności 4-krotnej jest prostsze. Wystarczy zliczyć liczbę sąsiadów. Poniżej zakładamy, że piksel jest oznaczony jako krawędź w macierzy wejściowej. W zależności od liczby sąsiadów mamy następujące przypadki:

- 0 — wierzchołek izolowany,
- 1 — punkty końcowe (ang. *endpixels*),
- 2 — punkty łączące (ang. *linkpixels*), czyli fragmenty krawędzi,
- 3–4 — punkty węzłowe (ang. *vertices*), czyli punkty, od których odchodzą co najmniej trzy krawędzie.

W przypadku łączności 8-krotnej do detekcji wierzchołków należałyby stosować przekształcenia Hit-or-Miss z elementami strukturalnymi. Elementy strukturalne do wykrywania odpowiednich punktów są następujące:

3. OPIS AUTORSKIEGO NARZĘDZIA INFORMATYCZNEGO

- wierzchołek izolowany:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

- punkty końcowe (ang. *endpixels*):

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ z & z & z \end{bmatrix},$$

- punkty łączące (ang. *linkpixels*), czyli fragmenty krawędzi, posiadają dokładnie dwóch sąsiadów, nie używamy przekształcenia Hit-or-Miss a tylko liczymy sąsiadów

- punkty węzlowe (ang. *vertices*), czyli punkty, od których odchodzą co najmniej trzy krawędzie:

$$\begin{bmatrix} z & 1 & z \\ z & 1 & z \\ z & z & 1 \end{bmatrix} \text{ lub } \begin{bmatrix} 1 & z & z \\ z & 1 & z \\ 1 & z & 1 \end{bmatrix},$$

Warto zauważyć, że te elementy strukturalne należy obracać o 90, 180 i 270 stopni. Za każdym razem trzeba by wielokrotnie sprawdzać te same piksele. Ponadto należy sprawdzać 8, a nie 4 sąsiadów.

Kolejnym problemem jest fakt, że przy spójności 8-krotnej przekształcenie Hit-or-Miss może w najbliższym otoczeniu punktu krzyżowania się krawędzi oznaczyć kilka otaczających punktów, jako punkty węzlowe. Jest to złe rozwiązanie, ponieważ w ten sposób może nawet kilkukrotnie zwiększyć liczbę wierzchołków w grafie, co przełożyłoby się na niską wydajność algorytmu.

Ostatnim problemem z jakim należałoby się liczyć wybierając łączność 8-krotną jest fakt, że macierz wejściową dla tego etapu algorytmu trzeba by poddać procesowi szkieletyzacji. Najlepiej byłoby w tym celu skorzystać z algorytmu KMM [22] lub K3M [23]. Te algorytmy musiałyby co najmniej raz przejrzeć całą macierz z wykrytymi krawędziami w optymistycznym przypadku.

W pracy zdecydowano się na metrykę 4-krotną ze względu na prostotę obliczeń, która skróciła czas pracy algorytmu. Nie wymagało to także dodatkowych obliczeń związanych z ścienianiem. Przygotowano i zaimplementowano algorytm tworzący graf z bitmapy, a jego pseudokod znajduje się w Załączniku 1.

Utworzony w ten sposób graf jest grafem nieskierowanym z wagami, gdzie wagi to liczba pikseli, czy też punktów należących do krawędzi. Graf ten może nie być spójny. Ponadto może nie zawierać wierzchołków, które pokrywają się z punktami wybranymi przez użytkownika.

3.4. MODUŁ OBRYSÓW PÓŁAUTOMATYCZNYCH

Graf ten zazwyczaj jest rzadki, ponieważ liczba jego krawędzi jest rzędu liczby jego wierzchołków. Z uwagi na czasami bardzo dużą liczbę wierzchołków — nawet do kilkudziesięciu tysięcy — próba implementacji przy pomocy macierzy sąsiedztwa mogłaby spowodować zużycie całej możliwej pamięci operacyjnej. Dla 50 tysięcy wierzchołków program musiałby zadeklarować macierz sąsiedztwa zawierającą 2,5 miliarda komórek. Z tych powodów graf został zaimplementowany przy pomocy list sąsiedztwa.

W przyjętej implementacji każda krawędź zawiera dodatkowo informację o tym, jakie piksele należą do danej krawędzi w rzeczywistym obrazie.

3.4.6. Zapewnienie spójności grafu

W pierwszym kroku na tym etapie przetwarzania grafu są dodawane punkty wybrane przez użytkownika do grafu.

Graf, który uzyskano w poprzednim kroku może nie być spójny. Powoduje to fakt, że między punktami wybranymi przez użytkownika mogą nie istnieć ścieżki. W celu zapewnienia spójności grafu należy dodawać sztuczne krawędzie. Zostają one dodawane z większymi wagami, niż wynikłoby to w rzeczywistości z liczebności listy pikseli, które reprezentują, ponieważ ma to na celu używanie z większym priorytetem prawdziwych krawędzi, a nie sztucznych. W sytuacji gdy nie istnieje dobra ścieżka z prawdziwych krawędzi, zostaną użyte krawędzie sztuczne. Duże wagi dodatkowo będą zmuszały algorytmy wyszukiwania najkrótszych ścieżek w grafie do minimalizowania długości takich fragmentów zawierających sztuczne krawędzie.

Do znajdowania spójnych składowych grafu najczęściej używa się algorytmu przeszukiwania grafu w głąb (ang. Depth-first search, w skrócie DFS) lub przeszukiwania grafu wszerz (ang. Breadth-first search, w skrócie BFS) [24]. Użyto algorytmu przeszukiwania wszerz, opierając się na przykładowej implementacji w materiałach [4]. Wykorzystano Queue<T>, a zatem kolejkę, więc jest to przeszukiwanie wszerz. Złożoność obliczeniowa tego algorytmu to $O(|E|)$.

Po wyznaczeniu spójnych składowych grafu dla każdej pary składowych znajdowano taką parę wierzchołków, żeby odległość między nimi jest minimalna. Jeśli odległość była mniejsza niż maksymalna odległość między dwoma dowolnymi punktami zaznaczonymi przez użytkownika, to była dodawana sztuczna krawędź o wadze 2,5 razy większej niż odległość wynikającą z metryki Manhattan pomiędzy tymi dwoma wierzchołkami.

Przykład zasady działania tej operacji przedstawia poniższy pseudokod:

```
foreach (Spójna składowa grafu s1)
{
    foreach (Spójna składowa grafu s2, różna od s1 i nie przetwarzana
```

3. OPIS AUTORSKIEGO NARZĘDZIA INFORMATYCZNEGO

```
wcześniej jako s1)
{
    Wybierz wierzchołek v1 z s1 i v2 z s2 takie, że odległość pomiędzy
    nimi jest najmniejsza ze wszystkich takich par
    Dodaj sztuczną krawędź pomiędzy v1 i v2
}
}
```

W ten sposób osiągnięto spójny graf, który zawiera punkty dodane przez użytkownika.

3.4.7. Wyszukanie najkrótszych ścieżek w grafie

Na tym etapie została zapewniona spójność grafu. Z całą pewnością istnieje ścieżka pomiędzy punktami wybranymi przez użytkownika, te punkty są osiągalne. Pozostało wybrać najkrótszą ścieżkę łączącą kolejne punkty. Założono, że jest dana lista punktów wybranych przez użytkownika i zawiera ona kolejne punkty, tzn. pierwszy należy połączyć z drugim, drugi z trzecim, . . . , ostatni z pierwszym.

Warto zaznaczyć, że wyszukiwano ścieżki o minimalnej wadze, czy też o minimalnym koszcie. Z uwagi na fakt, że wagi w grafie są ściśle związane z odległościami to używane jest sformułowanie szukania najkrótszych ścieżek. Wagi w tym grafie są nieujemne. Może w nim istnieć kilka ścieżek z jednego punktu do drugiego o tym samym koszcie, więc algorytm kończy swoje działanie na tym etapie, gdy znajdzie jedną z nich. W tym grafie nie występują krawędzie wielokrotne i nie zawiera cykli własnych.

Do wyszukiwania najkrótszych ścieżek w grafie, po zgłębieniu literatury [24] i [4] były rozważane do użycia 2 algorytmy. Był to algorytm Dijkstry i A*. Do algorytmu A* była rozważana heurystyka w postaci odległości miejskiej, Manhattan z wierzchołka v do celu t , ozn. $h(v)$.

Zgodnie z [4] „Funkcja h musi spełniać następujące warunki:

- musi być oszacowaniem dolnym, czyli dla każdego wierzchołka v $h(v) \leq$ odległość v od celu t ,
- musi być monotoniczna, czyli dla dowolnej krawędzi $\langle u, v \rangle$ $h(u) \leq h(v)$.

Heurystyka w postaci metryki Manhattan spełnia te wymagania. Z tego powodu może zostać wykorzystany algorytm A*. Porównując złożoności algorytmu A* i Dijkstry w [4] możemy zauważać, że algorytm A* w pesymistycznym przypadku ma taką samą złożoność jak algorytm

3.4. MODUŁ OBRYSÓW PÓŁAUTOMATYCZNYCH

Dijkstry, czyli $O(|E| * \log(|V|))$ z kolejką priorytetową dla grafów rzadkich, a $O(|V|^2)$ nie wykorzystując kolejki priorytetowej dla grafów gęstych. W praktyce dla grafów rzadkich nie ma potrzeby rozważania znacznej części wierzchołków, co poprawia złożoność średnią. Wynosi ona wtedy $O(|E|)$.

Do implementacji wyszukiwania najkrótszych ścieżek w grafie na podstawie wyżej wymienionych wniosków został wykorzystany algorytm A* z heurystyką w postaci metryki Manhattan. Przykładowy pseudokod tego algorytmu można znaleźć w [4] i jest on następujący:

```

CLOSE = 0 // zbiór (z szybkim sprawdzeniem przynależności)
OPEN = {s} // kolejka priorytetowa
// priorytety - sumy odległości wierzchołków od źródła
// i oszacowań odległości tych wierzchołków od celu
odległość[s] = 0
while ( OPEN niepusty )
{
    u = wierzchołek należący do OPEN taki, że
        odległość[u] + oszacowanie[u,t] <=
        odległość[w] + oszacowanie[w,t]
        dla wszystkich w należących do OPEN
    usuń u z OPEN
    wstaw u do CLOSE
    if ( u == t ) break
    foreach ( wierzchołek w sąsiadujący z u taki, że w należący do CLOSE )
    {
        if ( w należy do OPEN )
        {
            odległość[w] = nieskończoność
            wstaw w do OPEN
        }
        if ( odległość[w] > odległość[u] + waga<u,w> )
        {
            odległość[w] = odległość[u] + waga<u,w>
            aktualizacja priorytetu wierzchołka w (w OPEN)
            poprzedni[w] = u
        }
    }
}

```

3. OPIS AUTORSKIEGO NARZĘDZIA INFORMATYCZNEGO

```
}
```

```
}
```

Aby maksymalnie dobrze wykorzystać niską złożoność obliczeniową algorytmu A* należało wykorzystać dobre struktury danych do tego algorytmu. Zbiór CLOSE wymagał szybkiego sprawdzania przynależności. Po przeanalizowaniu rekomendowanych struktur danych [13] został użyty HashSet<T>.

Zbiór Open oferował zastosowanie znacznie bardziej finezyjnych struktur danych. Wymagane było od niego, aby był kolejką priorytetową, czyli aby był sortowany po priorytetach będącymi sumą odległości wierzchołków od źródła i oszacowań odległości tych wierzchołków od celu. Warto zauważyc, że oprócz sortowania często są wstawiane do niego wierzchołki, pobierane, usuwane i modyfikowane wartości priorytetu. Bardzo często jest sprawdzana przynależność i wyszukiwanie według klucza.

Porównanie czasów różnych operacji dla różnych klas słownikowych zaimplementowanych w technologii .NET można znaleźć w [2]. Najlepszym rozwiązaniem byłoby użycie SortedDictionary<K,V>, która jest implementowana przez drzewo czerwono-czarne. Niestety, jest sortowana po kluczu, a nie tak jak jest potrzebne w tym przypadku sortowanie po wartości. Poszukiwano zatem struktury danych, która sortuje po wartościach i ma łatwy dostęp przez klucz.

W naszym rozwiążaniu została zaimplementowana struktura danych opierająca się na dwóch podstrukturach - zaimplementowanej kolejki priorytetowej poprzez listę, oraz słownik Dictionary<K,V> z technologii .NET, który opiera się o Tablicę skrótów. Zaimplementowana lista miała następującą strukturę:

```
public class MySortedListElement
{
    public Vertex Key;
    public double Value;
    public MySortedListElement next;
    public MySortedListElement previous;
}
```

Struktura ta miała zaimplementowane sortowanie przez wstawianie, zatem modyfikacja tylko jednego elementu wymagała w pesymistycznym przypadku tylko $O(n)$ porównań. Wstawianie nowego elementu ma złożoność pesymistyczną $O(n)$, usuwanie $O(1)$. Zbadanie przynależności po kluczu to $O(n)$ i nie zaleca się tego robić.

Niezależnie od tej struktury jest przechowywana druga struktura danych, Dictionary<K,V>,

3.4. MODUŁ OBRYSÓW PÓŁAUTOMATYCZNYCH

gdzie kluczami są wierzchołki, a wartości to referencje na elementy tej listy. Zmienne odpowiadające za przechowanie kolejki priorytetowej OPEN zostały zadeklarowane w sposób następujący:

```
Dictionary<Vertex, MySortedListElement> OpenDictionary =  
    new Dictionary<Vertex, MySortedListElement>();  
  
MySortedList OpenList = new MySortedList();
```

W ten sposób w połączeniu tych dwóch podstruktur otrzymujemy strukturę danych o następujących właściwościach:

- dodawanie w czasie $O(n)$,
- wyszukiwanie po kluczu w czasie $O(1)$,
- wybieranie najmniejszy element według wartości w czasie $O(1)$,
- usuwanie w czasie $O(1)$
- modyfikowanie jednego elementu w czasie $O(n)$.

Algorytm A* jest uruchamiany dla każdego punktu wybranego przez użytkownika. Wyszukuje on najkrótszą ścieżkę do kolejnego punktu wybranego przez użytkownika. Po wszystkich obliczeniach obrys składa się z poszczególnych ścieżek. Są one konsolidowane i zwracane jako lista pikseli na podstawie danych z krawędzi o listach pikseli, z których jest zbudowana krawędź. W ten sposób jest wykrywany obrys pomiędzy punktami zaznaczonymi przez użytkownika.

W sytuacji, gdy pomiędzy różnymi wierzchołkami nie istnieją prawdziwe krawędzie, albo istnieją tylko w wybranym fragmencie ścieżki łączącej te dwa wierzchołki, algorytm w miarę możliwości będzie starał się używać prawdziwych krawędzi, dzięki odpowiedniej wadze krawędzi sztucznych. Dzięki sztucznym krawędziom na pewno istnieje droga między kolejnymi punktami, zatem algorytm z całą pewnością zwróci poprawny wynik. Co najwyżej będzie to wielokąt z punktami wybranymi przez użytkownika.

Algorytm dla sztucznych krawędzi generuje listę pikseli algorytmem Bresenhama [3]. Jest to algorytm, który dodaje w najbardziej optymalny sposób krawędzie. W celu zapewnienia łączności 4-krotnej jest przeprowadzana operacja morfologiczna z wykrywaniem 2 pikseli po przekątnej z maską 4-pikselową. Gdy takie piksele zostaną wykryte, to na jednym z rogów jest dodawany piksel w celu zapewnienia łączności 4-krotnej.

3.4.8. Optymalizacja algorytmu obrysu półautomatycznego

Większość obrazów medycznych ma rozdzielcość rzędu kilkaset na kilkaset pikseli, a więc na całym obrazie znajduje się kilkaset tysięcy pikseli. Zdarzają się też pliki w znacznie większej

3. OPIS AUTORSKIEGO NARZĘDZIA INFORMATYCZNEGO

rozdzielczości, takimi są między innymi mammografia i zdjęcie rentgenowskie rzędu kilku tysięcy na kilka tysięcy pikseli. Na tych obrazach znajduje się do kilkudziesięciu milionów pikseli. W celu sprawnego przetwarzania tych obrazów potrzebne było przeprowadzenie optymalizacji algorytmu. Zostały dodane usprawnienia, które opisano poniżej.

1. Zmniejszenie rozmiaru obrazu roboczego

W pierwszym kroku zmniejszono rozmiar obrazu roboczego. Wyznaczono najmniejszy obszar zainteresowania w kształcie prostokąta, w którym znajdują się wszystkie punkty zaznaczone przez użytkownika. Powiększono ten obszar o marginesy w taki sposób, by wyjściowy prostokąt nadal był w środku, a pole powiększonego było 4-krotnie większe. Innymi słowami — dodano marginesy po 50% szerokości/wysokości do każdego wymiaru. Jeśli rozmiar powiększonego prostokąta jest większy niż obrazu, to nie jest zmieniany obraz roboczy. Gdy użytkownik stworzył niewielki obrys, to rozmiar obrazu jest także niewielki. W sytuacji, gdy obrys zajmuje prawie cały obraz medyczny, to nadal musi być przetwarzany cały obraz.

2. Sięganie do pamięci zamiast to bitmapy

W celu przyśpieszenia działania operatora Sobela i liczenia statystyk, zamiast sięgać do bitmapy metodą `.GetPixel()` na platformie .NET, bitmapa została zapisana do tablicy bajtów w celu uzyskania bezpośredniego dostępu. Została wykorzystana do tego funkcja `.LockBits()`. Zysk na tej operacji był kilkudziesięciokrotny. Na późniejszych etapach algorytmu pracowano na macierzy, która nie była już bitmapą, więc operacje te wykonywały się znacznie szybciej.

3. Dzielenie zbyt długich krawędzi

W sytuacji, gdy wczytane krawędzie do grafu są bardzo długie, może zdarzyć się taka sytuacja, że punkt zaznaczony przez użytkownika znajduje się bardzo blisko krawędzi, ale daleko od punktu początkowego lub końcowego krawędzi. W tym celu jest wprowadzony mechanizm dzielenia zbyt długich krawędzi na kilka krótszych.

4. Problem z ilością punktów

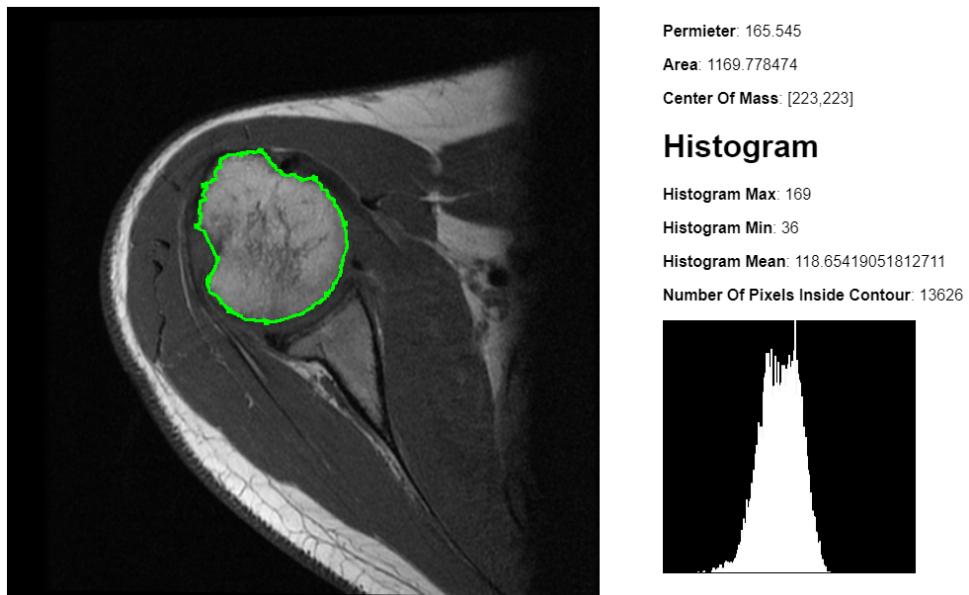
Dla każdego punktu jest uruchamiany algorytm A*, zatem w sposób liniowy od ilości punktów zależy liczba uruchomień algorytmu A*. Nie można wyznaczyć, czy w złożoności średniej całego algorytmu algorytm jest zależny liniowo od ilości punktów, czy w sposób logarytmiczny, czy też w sposób stały.

3.5. Moduł obliczeń statystyk

System zapewnia liczenie statystyk dotyczących obrysu. Statystyki są identyczne zarówno dla obrysów manualnych i półautomatycznych, do których należą:

- histogram — wykres pokazujący natężenie kolorów w obrysowanym obszarze (kolor, to liczba w zakresie [0, 255], gdzie 0 oznacza najciemniejszy piksel, a 255 najjaśniejszy),
 - wartość maksymalna — maksymalna wartość koloru piksela wewnątrz obrysu,
 - wartość minimalna — minimalna wartość koloru piksela wewnątrz obrysu,
 - wartość średnia — średnia arytmetyczna wartości kolorów pikseli wewnątrz obrysu,
- środek ciężkości — średnia arytmetyczna pozycji pikseli obrysu,
- długość obrysu w mm — obliczana na podstawie położenia pikseli i wartości pixel spacing odczytywanej z tagu obrazu DICOM,
- długość obrysu w pikselach,
- pole powierzchni w mm^2 ,
- liczba pikseli wewnątrz obrysu.

Przykładowe wygenerowane statystyki zostały przedstawione na Rysunku 3.7.



Rysunek 3.7: Przykładowe wygenerowane statystyki

Do obliczenia histogramu jest wymagany punkt wewnętrzny obrysu. Środek ciężkości obrysu nie zawsze musi znajdować się wewnątrz obrysu. W przypadku zastosowania algorytmu scan-linii dla obrysu manualnego nie jest podana informacja o logicznych wartościach, gdzie jest krawędź, więc nie można wykryć np. krawędzi poziomych.

3. OPIS AUTORSKIEGO NARZĘDZIA INFORMATYCZNEGO

Z tego powodu wybrano algorytm Flood Fill [4], czyli rozlewania się rekurencyjnego zliczonych pikseli od punktu wewnętrznego. Właśnie z tego powodu od użytkownika jest wymagane podanie dodatkowej informacji, jaką jest punkt wewnętrzny obrysu. Założono, że użytkownik zaznaczy go poprawnie. W sytuacji, gdy zostanie podany błędny punkt to ten algorytm policzy to co znajduje się na zewnątrz obrysu, a nie to co jest wewnętrz.

Długość obrysu w pikselach jest obliczana na podstawie liczebności listy pikseli należących do obrysu.

3.6. Moduł anonimizacji danych

W celu umożliwienia anonimizacji danych pacjentów zawartych w plikach DICOM skorzystano z REST API serwera Orthanc. Umożliwia ono wykonanie kopii badania wgranego do serwera z nowymi danymi pacjenta. Po wykonaniu kopii z podanymi przez użytkownika danymi oryginalny obraz DICOM jest usuwany z serwera obrazów Orthanc.

4. Przeprowadzone eksperymenty

Zostały przeprowadzne eksperymenty na autorskim narzędziu do tworzenia obrysów. W podrozdziałach 4.1 - 4.4 przedstawiono wyniki tych eksperymentów.

Testy wykonano na komputerze stacjonarnym o następującej specyfikacji:

- procesor Intel Core i5-7500 o taktowaniu maksymalnym dla jednego rdzenia 3.8 Ghz,
- pamięć RAM Corsair Vengeance DDR4 16GB 3000 Mhz ustawiona w tryb 2140 Mhz o opóźnieniach CL15,
- płyta główna ASUS STRIX ROG Z270-I,
- procesor jest chłodzony powietrzem przez Noctuę NH-L9i, procesor utrzymuje temperaturę około 64 stopni Celsjusza przy temperaturze otoczenia około 21 stopni Celsjusza, nie występuje throttling,
- system operacyjny Windows 10.

4.1. Zbiór testowy

Na zbiór plików testowych składają się prywatne badania autora i jego najbliższej rodziny.

Do testów zostały wykorzystane następujące badania:

1. badanie rezonansem magnetycznym barku prawego Tomasza Świerczewskiego około 8 tygodni po zwichtnięciu stawu ramiennego typu przedniego podkruczego — Badanie 1,
2. zdjęcie rentgenowskie barku prawego Tomasza Świerczewskiego w momencie zwichtnięcia stawu ramiennego — Badanie 2,
3. badanie tomografią komputerową kręgosłupa Wojciecha Świerczewskiego — Badanie 3,
4. badanie rezonansem magnetycznym kręgosłupa Wojciecha Świerczewskiego — Badanie 4.

Pliki te były wykonane na różnych urządzeniach, w różnych szpitalach.

Dodatkowo wykorzystano wyniki badań udostępnione Politechnice Warszawskiej do celów dydaktyczno badawczych. Zawierały one badania organów wewnętrznych, takich jak wątroba i trzustka wykonane przy użyciu rezonansu magnetycznego oraz badanie mammograficzne.

4.2. Analiza działania aplikacji

W autorskiej aplikacji otwierano kolejne badania. Rysunek 4.1 przedstawia jedną klatkę obrazu medycznego Badanie 1, z pewnej serii. Rysunek 4.2 przedstawia Badanie 2, natomiast Rysunek 4.3 przedstawia Badanie 3, a Rysunek 4.4 Badanie 4.

Ten test wykazał, że aplikacja otwiera różne badania medyczne, od rezonansu magnetycznego, przez tomografię komputerową po zdjęcie rentgenowskie. Na każdym z obrazów medycznych można wykonywać obrysy. Przykładowe obrysy ręczne i półautomatyczne znajdują się odpowiednio na Rysunku 4.5 i Rysunku 4.6. Wyniki testów potwierdzają, że statystyki są obliczane poprawnie.

4.3. Wydajność algorytmu półautomatycznego

W celu sprawdzenia wydajności stworzonego algorytmu półautomatycznego posłużono się Badaniem 1, o rozdzielczości 512 na 512 pikseli.

W pierwszym teście sprawdzono czasy generowania obrysu półautomatycznego kości ramiennej, w zależności od ilości punktów wybranych przez użytkownika. Oczekiwano, że obrys będzie wyglądał podobnie do tych na Rysunkach 4.5 i 4.6. Rozmiary przetwarzanych wycinków obrazu w algorytmie były porównywalne. Algorytm wykorzystywał wycinki obrazu medycznego zawierające wszystkie punkty wybrane przez użytkownika z uwzględnieniem marginesów.

W trakcie dodawania kolejnych punktów starano się poprawiać obrys, tzn. aby wygenerowany obrys jak najlepiej odwzorowywał kość. Przy okazji dodawania kolejnych punktów do tego samego obrysu przetestowano funkcjonalność dodawania kolejnych punktów do obrysu.

Dla każdej liczby wierzchołków przeprowadzano 10 kolejnych pomiarów generowania tego samego obrysu. Pozwoliło to na wyznaczenie średniego czasu generowania obrysu, niezależnie od chwilowego wykorzystania procesora.

Większość z generowanych obrysów było podglądami, czyli nie obliczano dla nich statystyk. Czas mierzono poprzez sprawdzanie czasu potrzebnego na wygenerowanie odpowiedzi przez API na otrzymane zapytanie.

Na Rysunku 4.7 przedstawiono wyniki przeprowadzonych testów. Wraz z dodawaniem kolejnych punktów zmniejszał się czas potrzebny na wygenerowanie obrysu przez serwer. Warto zauważyć, że wartości zmniejszyły się o około 150 ms. Nie dodawano kolejnych punktów z uwagi na osiągnięcie zadowalającej jakości wygenerowanego obrysu.

Na Rysunku 4.8 przedstawiono wyniki dla drugiego z testów. Starano się obrysować cały

4. PRZEPROWADZONE EKSPERYMENTY

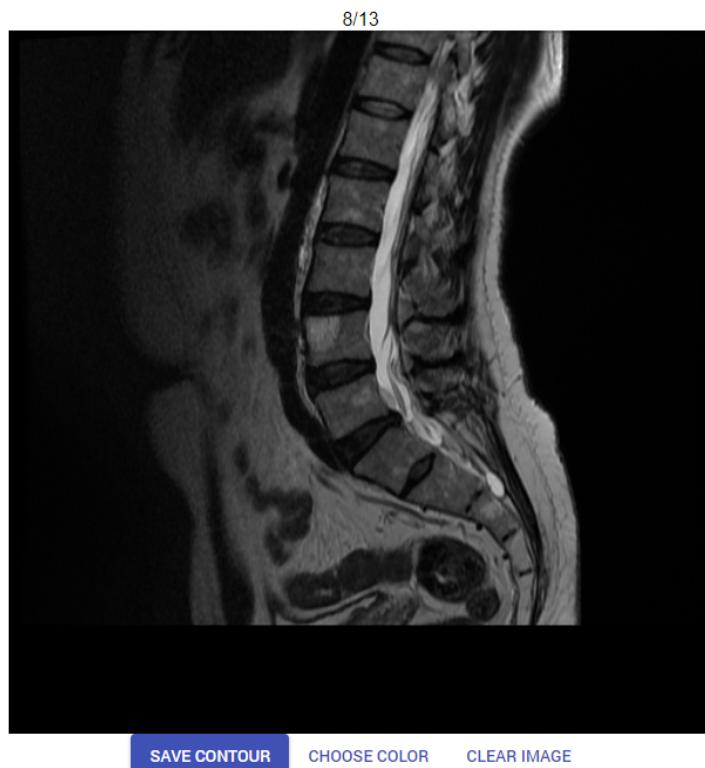


Rysunek 4.1: Widok badania rezonansem magnetycznym barku

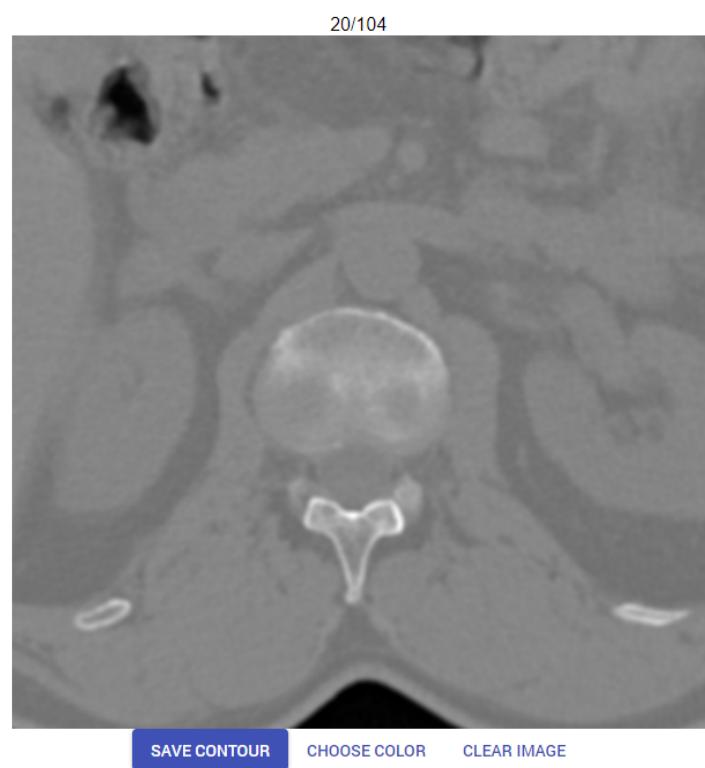


Rysunek 4.2: Widok zdjęcia rentgenowskiego barku

4.3. WYDAJNOŚĆ ALGORYTMU PÓŁAUTOMATYCZNEGO

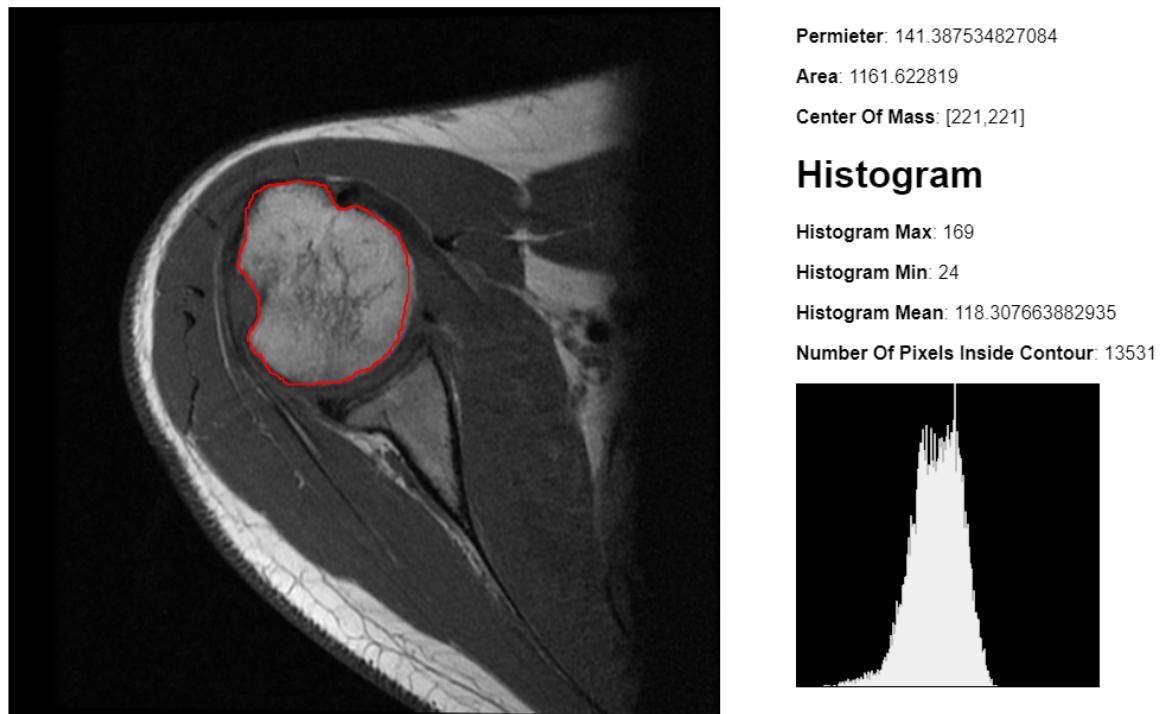


Rysunek 4.3: Widok badania rezonansem magnetycznym kręgosłupa

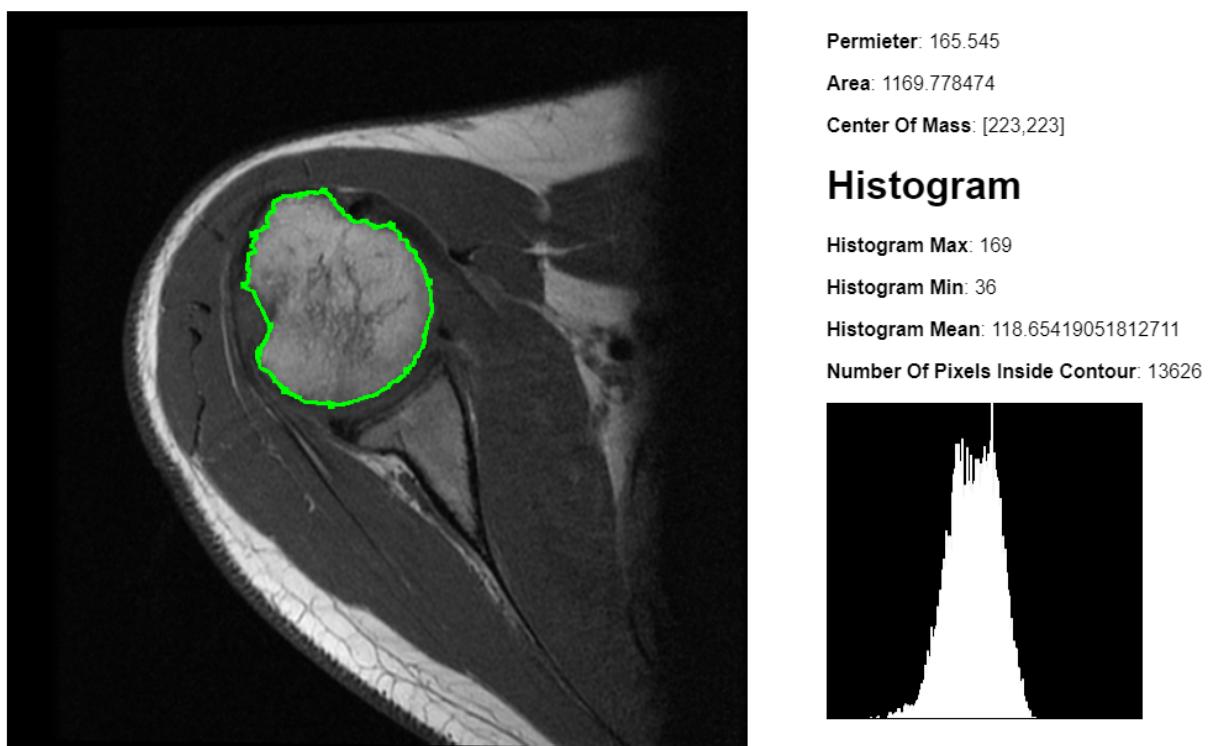


Rysunek 4.4: Widok badania tomografią komputerową kręgosłupa

Testowy ręczny

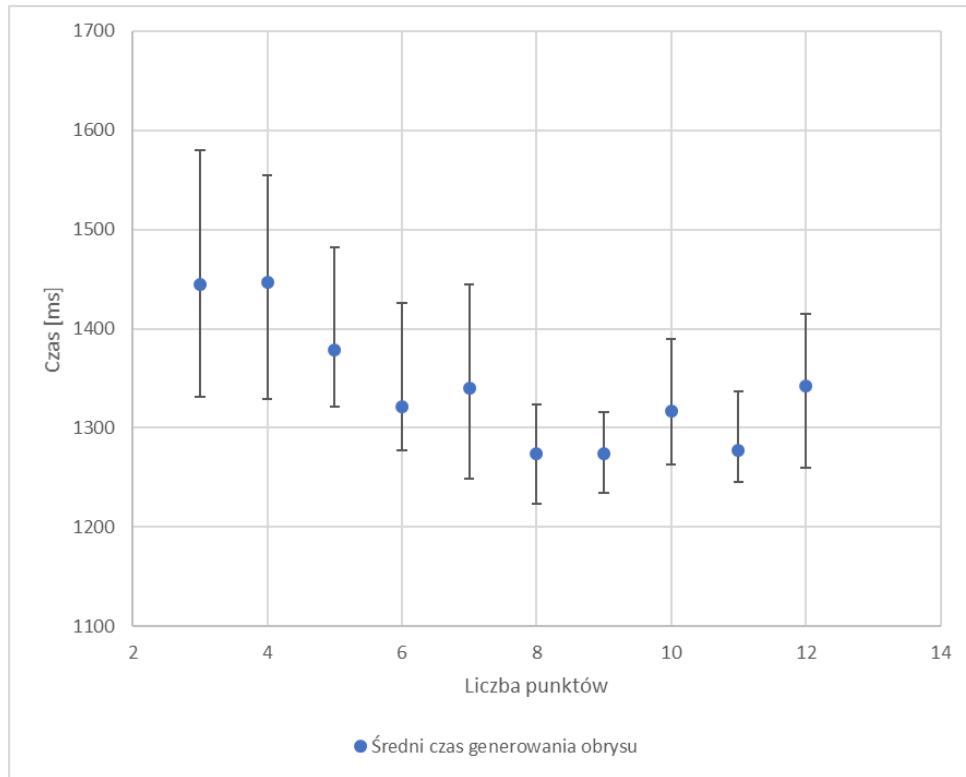


Rysunek 4.5: Przykładowy obrys manualny

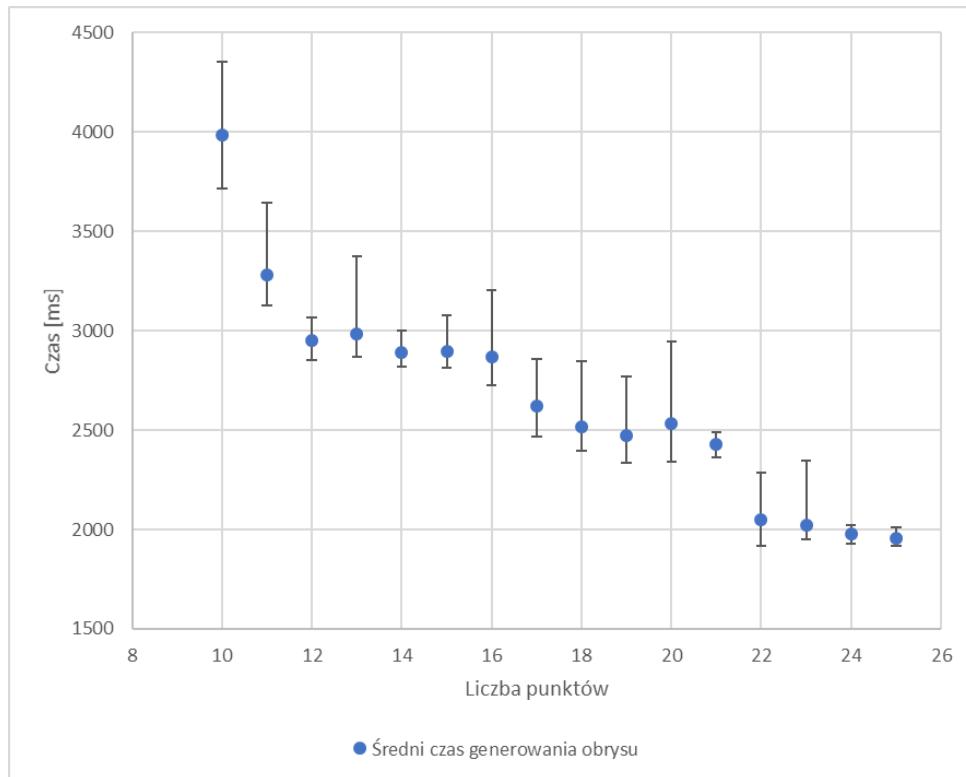


Rysunek 4.6: Przykładowy obrys półautomatyczny

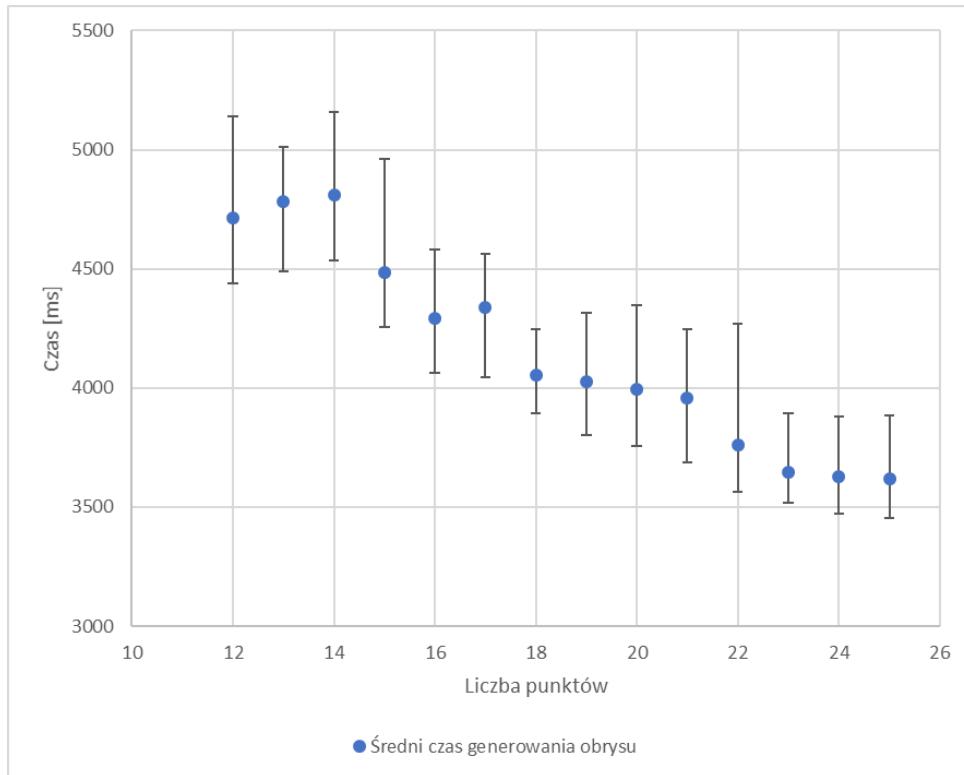
4.3. WYDAJNOŚĆ ALGORYTMU PÓŁAUTOMATYCZNEGO



Rysunek 4.7: Czasy generowania obrysu półautomatycznego kości ramiennej w zależności od ilości punktów



Rysunek 4.8: Czasy generowania obrysu półautomatycznego barku w zależności od ilości punktów



Rysunek 4.9: Czasy generowania obrysu półautomatycznego barku w zależności od ilości punktów drugą metodą

brak widoczny na obrazie medycznym. W tym celu wstawiano kolejne punkty i obserwowało się, jakie były potrzebne na wygenerowanie obrysu. Wraz z kolejnymi punktami poprawiano generowany obrys. Wygenerowany obrys zaprezentowano na Rysunku 4.10. Czasy generowania zmalały prawie 2 krotnie, z poziomu około 4 s do około 2 s.

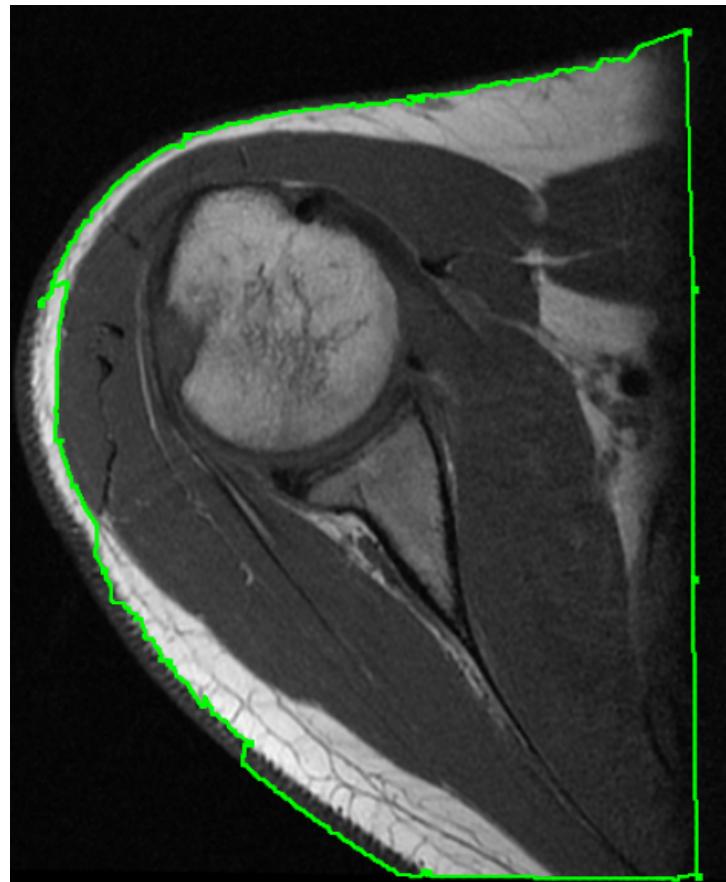
Na Rysunku 4.8 można zauważać czasami gwałtowne skoki w zmniejszającym się czasie potrzebnym na obliczenia. Mogło to być spowodowane faktem, że zmieniała się maksymalna odległość między kolejnymi punktami. Bliżej to zjawisko zostało opisane w podrozdziale 4.4.

Na Rysunku 4.9 przedstawiono wyniki trzeciego testu. Nie ingerowano we wspomnianą w podrozdziale 4.4 maksymalną odległość — wstawiano gęsto kolejne punkty obok siebie na łuku. W ten sposób sprawdzano wpływ liczby uruchomień algorytmu A*, poprzez zmianę liczby punktów, na czas obliczeń.

Uzyskane wyniki są bardzo podobne do poprzednich, choć bez wyraźnych skoków. Czas potrzebny na wygenerowanie obrysu nie spadł tak bardzo jak w drugim teście, ale spadek był zauważalny, z około 4,7 s do 3,7 s.

Wraz ze zwiększeniem się przestrzeni roboczej obrazu rośnie czas potrzebny na wygenerowanie obrysu. Na podstawie wykresów zależności czasu od liczby punktów można wysnuć wnioski, które zostaną zaprezentowane w podrozdziale 4.4.

4.4. ANALIZA WYNIKÓW I WNIOSKI



Rysunek 4.10: Jeden z wygenerowanych obrysów w trakcie testów

4.4. Analiza wyników i wnioski

Poniżej przedstawiono analizę wyników oraz wnioski z przeprowadzonych testów.

Zauważono, że gdy generujemy statystyki, to czas potrzebny na obliczenie statystyk zwiększał czas na realizację zapytania o około 30%. Wiązało się to z odwiedzeniem wszystkich punktów wewnętrz obrysu, co powodowało wydłużenie obliczeń.

Wraz ze zwiększaniem rozdzielczości zdjęcia, jak również ze zwiększeniem się obrysu na obrazach medycznych o wysokiej rozdzielczości, wzrastał proporcjonalnie czas potrzebny na wygenerowanie obrysu półautomatycznego. Dzieje się tak dlatego, że w algorytmie wykorzystano operator Sobela i Canny'ego, które wymagają odwiedzenia pewnej liczby pikseli, która odpowiednio się zwiększa. Dla mammografii średniej wielkości obrys był generowany nawet kilkadesiąt sekund. Dla większości obrysów czasy obliczeń nie przekraczały zakładanych 30 sekund w wymaganiach niefunkcjonalnych.

Wraz ze zwiększaniem liczby punktów zmniejszał się czas potrzebny na wygenerowanie obrysu półautomatycznego. Powodowały to dwa oddzielne zjawiska. Pierwsze z nich, to maksymalna

odległość pomiędzy kolejnymi dwoma punktami wybranymi przez użytkownika. W celu zapewnienia spójności grafu dodawano sztuczne krawędzie, o maksymalnej długości nieprzekraczającej maksymalnej odległości pomiędzy wcześniej wspomnianymi punktami. Im krótsza była to odległość, tym mniej było dodawanych sztucznych krawędzi. Dodawanie nowych krawędzi pomiędzy oddalonymi od siebie punktami oznaczało kilkukrotnie mniejszą liczbę przetwarzanych krawędzi przetwarzanych przez algorytm A*. Drugim zjawiskiem odpowiedzialnym za zmniejszanie się czasu potrzebnego na wygenerowanie obrysu półautomatycznego, wraz z zwiększającą się liczbą punktów, był algorytm A*. Początkowo przypuszczano, że zwiększenie liczby uruchomień algorytmu A* wraz ze wzrostem liczby punktów będzie generował dodatkowy koszt obliczeniowy, a nie go redukował. Tak może się stać w pesymistycznym przypadku. W realnych obrysach wraz z dodawaniem kolejnych punktów w celu poprawienia jakości obrysu małała złożoność średnia, ponieważ algorytm A* szybciej znajdował prawidłową ścieżkę. Algorytm A* dzięki heurystyce pomijał bardzo dużą liczbę krawędzi, które na pewno nie polepszyłyby rozwiązania.

Wraz ze wzrostem liczby punktów wybranych przez użytkownika najczęściej malał czas potrzebny na wygenerowanie obrysu półautomatycznego. Czas dla większych obrysów maleje w sposób znaczący, dla małych obrysów w sposób niezauważalny. Dzieje się tak dlatego, że operacje na grafach są tylko elementem obliczeń. W przypadku małych obrysów, dominującą częścią obliczeń są operacje na przetwarzanym obrazie. Ponadto czasy obliczeń zawierają narut czasowy wywołany przez przetwarzanie zapytań.

Na podstawie przeprowadzonych obserwacji zaleca się, aby użytkownicy unikali dużych obrysów, szczególnie na zdjęciach o wysokiej rozdzielczości rozdzielczości, czyli np. mammografiach albo zdjęciach rentgenowskich. Ponadto generowanie obrysów półautomatycznych działa zauważalnie lepiej dla zdjęć o ostrych krawędziach i o niskim szumie. W celu poprawienia zarówno jakości wykrywanego obrysu jak i wydajności czasowej przeprowadzanych w trakcie generowania obrysu obliczeń użytkownik powinien wybierać liczbę punktów rzędu kilkunastu punktów, co może mieć zauważalnie lepszą wydajność niż wybranie kilku punktów.

5. Podsumowanie

Cel pracy został osiągnięty. Spełniono wszystkie wymagania funkcjonalne i niefunkcjonalne, z wyłączeniem wymagania dotyczącego wydajności generowania obrysu półautomatycznego dla obrazów DICOM o dużej rozdzielczości (ponad 1000000 pikseli).

W trakcie pracy nad systemem napotkano różne problemy i ograniczenia, przede wszystkim związane ze specyfikacją obrazowych badań medycznych.

W interfejsie ograniczeniem okazał się dostępny rozmiar ekranu. Projektowanie aplikacji zakładało wykorzystanie ekranu o rozdzielczości 1920x1080, a więc ekranu panoramicznego. Niestety obrazy DICOM mają bardzo zróżnicowane układy — są obrazy kwadratowe, pionowe oraz poziome. Założenie, że ekran użytkownika jest ekranem 1920x1080 sprawia, że obrazy pionowe będą wyświetlane jedynie w niewielkiej części obszaru przeznaczonego dla obrazów. Jest to niestety ograniczenie, którego nie da się zlikwidować, gdyż przy założeniu rozdzielczości 1080x1920 powoduje analogiczny problem z obrazami poziomymi. Zaleca się skonfigurowanie aplikacji ze względu na rodzaj badań jakie będą występowały w systemie (MRI, RTG, MMG, itd.).

Podstawowym problemem występującym w trakcie implementacji rozwiązania było ustalanie kontraktów w warstwie komunikacyjnej. Ze względu na niewielką ilość akcji w komunikacji nie zdecydowano się na zastosowanie generatora kontraktów, ale zaleca się wprowadzenie takiego rozwiązania ponownie w trakcie dalszego rozwoju narzędzia. W zależności od złożoności komunikacji generator kontraktów może zdecydowanie usprawnić wprowadzanie zmian w aplikacji.

Podczas projektowania systemu zdecydowano, że obrazy DICOM będą wyświetlane w największej rozdzielczości umożliwiającej wyświetlenie całego obrazu na ekranie. W związku z tym, większość obrazów wyświetlona jest w rozmiarze różnym od faktycznego rozmiaru obrazu. Rozważano dwie możliwości rozdzielczości wykonywanych obrysów: realne wymiary obrazu oraz rozmiary obrazu wyświetlonego na ekranie. Zdecydowano, że obrys powinny być zapisywane w wymiarach identycznych realnym rozmiarom obrazu. Decyzja została uzasadniona potrzebą zapewnienia poprawnego obliczania statystyk. Zapisywanie obrysów w takich wymiarach ułatwia obliczanie liczby pikseli wewnątrz obrysu.

Ze względu na złożoność algorytmu używanego do wyznaczania obrysu półautomatycznego wykonywanie obrysów półautomatycznych w obrazach o dużej rozdzielczości przekracza dwukrot-

nie dopuszczalny czas 30 sekund. Czas obliczeń można poprawić poprzez ograniczenie obszaru, w którym wykrywane będą krawędzie, ale nie rozwiąże to problemu z wykonywaniem obrysów o wymiarach zbliżonych do pełnych wymiarów obrazu.

Podczas pracy nad narzędziem rozważano dodatkowe funkcjonalności, które nie zostały poruszone w tej pracy. Są one potencjalnymi możliwościami rozwoju narzędzia stworzonego w ramach tej pracy. Funkcjonalności opisano szczegółowo poniżej.

W obecnej wersji systemu nie zaimplementowano deskryptorów kształtu obrysu. Zaimplementowanie takich deskryptorów mogłoby dostarczyć dodatkowych informacji na temat wykonanych przez użytkownika obrysów. Taka informacja może w przyszłości dostarczyć dodatkową zmienną, którą można by wykorzystywać w celu trenowania sztucznej inteligencji w zautomatyzowanym wykrywaniu organów, jak również wykrywaniu i sugerowaniu niepokojących zmian.

W tej wersji w systemie użytkownicy nie są rozróżnialni. Dodanie użytkowników pozwoliłoby na segregowanie tworzonych obrysów i wyświetlanie użytkownikowi obrysów wykonanych jedynie przez niego samego, a nie wszystkich obrysów istniejących w systemie. Z punktu widzenia użyteczności systemu użytkownik nie musiałby szukać swojego obrysu pośród obrysów innych użytkowników systemu.

Uwierzytelnianie zapobiegłoby również niepowołanemu dostępowi osób postronnych do wykonanych obrysów oraz uniemożliwiłoby ataki typu DoS. W obecnej wersji można poprzez wysyłanie dużej liczby zapytań związanych z generowaniem obrysów półautomatycznych doprowadzić do niedostępności przeprowadzania akcji na serwerze. System jest również podatny na złośliwe działanie mające na celu zapełnienie całej dostępnej serwerowi przestrzeni dyskowej, które może zostać wywołane przez wysłanie dużej liczby zapisów obrysów manualnych.

Z punktu widzenia użytkownika interesujące mogą być informacje wyliczone przez system na temat wykonanych przez niego obrysów. W obecnej wersji systemu, jeśli użytkownik chciałby takie informacje zapisać musiałby samodzielnie przepisać dane wyświetlane w widoku szczegółów obrysu. Zautomatyzowanie takiej funkcjonalności mogłoby zaoszczędzić użytkownikowi wiele czasu.

Głównym celem wykonywania obrysów na obrazach medycznym jest generowanie zbioru testowego dla różnych metod sztucznej inteligencji mających na celu sugerowanie lekarzom niepokojących zmian wykrytych automatycznie. W związku z tym użytecznym rozszerzeniem funkcjonalności systemu byłoby zintegrowanie go z modułem sztucznej inteligencji i umożliwienie obrysowywania wgrywanych obrazów metodami sztucznej inteligencji opartych na obrysach wykonanych przez użytkownika.

Bibliografia

- [1] Abramov D. and the Redux documentation authors: ReduxJs <https://redux.js.org/> [Dostęp 23 lutego 2019]
- [2] Albahari J., Albahari B.: C 6.0 w pigułce, *Helion, O'Reilly Media, Inc.*, Gliwice, page–page, 2016
- [3] Bresenham J. E.: Algorithm for computer control of a digital plotter. *ICM System Journal* 4(1) 1965
- [4] Bródka J.: Wykłady z przedmiotu Algorytmy i Struktury Danych 2 *Politechnika Warszawska, Wydział Matematyki i Nauk Informacyjnych* Materiały dostepne na stronie: <http://mini.pw.edu.pl/brodka/ASD2.html> [Dostęp 22 stycznia 2019]
- [5] Canny J. F.: Finding Edges and Lines in Images. *Technical report no. 720, Massachusetts Institute of Technology (MIT)*, Cambridge, Massachusetts, USA, 1983
- [6] Cytowski J., Gielecki J., Gola A.: Cyfrowe przetwarzanie obrazów medycznych: Algorytmy. Technologie. Zastosowania. *Akademicka Oficyna Wydawnicza EXIT*, Warszawa, 88–94, 2008
- [7] Facebook Inc.: Informacje o bibliotece ReactJS. Oficjalna strona: <https://reactjs.org/> [Dostęp 27 stycznia 2019]
- [8] Hafey C.: Dokumentacja projektu Cornerstone Core. Oficjalna strona: <https://docs.cornerstonejs.org/> [Dostęp 27 stycznia 2019]
- [9] Hart P. E., Nilsson N. J., Raphael B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths *IEEE Transactions on Systems Science and Cybernetics* 4(2), 100–107, 1968
- [10] ivmartel: Projekt DICOM Web Viewer. Oficjalna strona: <https://ivmartel.github.io/dwv/> [Dostęp 27 stycznia 2019]

BIBLIOGRAFIA

- [11] Kronis K., Uhanova M.: Performance Comparision of Java EE and ASP.NET Core Technologies for Web API Developmnet. *Applied Computer Systems 23, Riga Technical University*, Ryga, 37–44, 2018
- [12] Microsoft Corporation: Dokumentacja platformy .NET. Oficjalna strona: <https://docs.microsoft.com/pl-pl/dotnet/> [Dostęp 22 stycznia 2019]
- [13] Microsoft Corporation: Dokumentacja rekomendowanych struktur danych platformy .NET. Oficjalna strona: <https://docs.microsoft.com/pl-pl/dotnet/standard/collections/> [Dostęp 22 stycznia 2019]
- [14] Microsoft Corporation: Informacje o platformie .NET Core. Oficialna strona: <https://docs.microsoft.com/pl-pl/dotnet/core/about> [Dostęp 22 stycznia 2019]
- [15] Microsoft Corporation: Informacje o platformie ASP.NET Core. Oficialna strona: <https://docs.microsoft.com/pl-pl/aspnet/core/?view=aspnetcore-2.2> [Dostęp 22 stycznia 2019]
- [16] Mozilla and individual contributors: Dokumentacja HTMLCanvasElement. Oficjalna strona: <https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement/> [Dostęp 27 stycznia 2019]
- [17] National Electrical Manufacturers Association: Standard DICOM. Oficjalna strona: <https://www.dicomstandard.org/> [Dostęp 22 stycznia 2019]
- [18] Nowacki R., Plechawska-Wójcik M.: Analiza porównawcza narzędzi do budowania aplikacji Single Page Application — AngularJS, ReactJS, Ember.js, *Journal of Computer Sciences Institute 2, Politechnika Lubelska, Instytut Informatyki*, Lublin, 98–103, 2016
- [19] Osimis S.A.: Projekt Orthanc. Oficjalna strona: <https://www.orthanc-server.com/> [Dostęp 27 stycznia 2019]
- [20] Osimis S.A.: Instrukcja korzystania z REST API Orthanc. Oficjalna strona: <http://book.orthanc-server.com/users/rest.html> [Dostęp 23 lutego 2019]
- [21] Rosenfeld A., Kak A. C.: Digital Picture Processing, *Academic Press, Inc.*, Nowy Jork, 1982
- [22] Saeed K., Rybnik M., Tabędzki M., Adamski M.: Algorytm do Ścieniania Obrazów: Implementacja i Zastosowania *Zeszyty Naukowe Politechniki Białostockiej 2002 Informatyka - Zeszyt 1*, Białystok, 2002

- [23] Saeed K., Tabędzki M., Rybnik M., Adamski M.: K3M: A Universal Algorithm for Image Skeletonization and a Review of Thinning Techniques *International Journal of Applied Mathematics and Computer Science*, 2010, 20(2) Białystok, 317–335, 2010
- [24] Sedgewick R., Wayne K.: Algorytmy Wydanie IV, *Helion*, Gliwice, 526–706, 2012
- [25] Sobel I., Feldman G.: An 3x3 Isotropic Image Gradient Operator for Image Processing. *Presentation at Stanford Aartificial Intelligence Project (SAIL) in 1968*, 2014
- [26] Weisstein, Eric W.: Moore Neighborhood. *From MathWorld—A Wolfram Web Resource*. <http://mathworld.wolfram.com/MooreNeighborhood.html> [Dostęp 22 stycznia 2019]

Wykaz symboli i skrótów

- API — ang. Application Programming Interface — zestaw ściśle określonych reguł, poprzez które komunikują się ze sobą programy
- CRUD — ang. Create, Read, Update, Delete — utwórz, odczytaj, aktualizuj i usuń — cztery podstawowe funkcje w aplikacjach, które umożliwiają zarządzanie nimi
- CSV — ang. Comma-Separated Values (w pracy użyte w kontekście formatu pliku)
- DoS — ang. Denial of Service — atak na system komputerowy mający na celu uniemożliwienie działania systemu
- DICOM — ang. Digital Imaging and Communications in Medicine — norma ujednolicająca wymianę i interpretację obrazowych danych medycznych
- HTML5 — ang. HyperText Markup Language 5 — język do tworzenia i prezentowania stron internetowych www
- HTTP — ang. Hypertext Transfer Protocol — protokół wymiany danych hipertekstowych
- HTTPS — ang. Hypertext Transfer Protocol Secure — szyfrowana wersja protokołu HTTP
- ID — ang. Identifier — unikalna nazwa przypisana do danego obiektu, pozwalająca rozróżnić obiekty w systemie
- JSON — ang. JavaScript Object Notation — lekki format wymiany danych, bazujący na podzbiorze języka JavaScript
- REST — ang. Representational State Transfer — styl architektury oprogramowania dla

WYKAZ SYMBOLI I SKRÓTÓW

systemów rozproszonych

- RGB — ang. Red, Green, Blue — model przestrzeni barw, opisanej współrzędnymi 3 barw podstawowych: czerwonej, zielonej i niebieskiej
- SDK — ang. Software Development Kit — zestaw narzędzi programistycznych niezbędny do tworzenia aplikacji korzystającej z danej biblioteki

Spis rysunków

2.1 Przykład użycia aplikacji OHIF Viewer — oznaczono: prostokątny obszar, dla którego obliczone zostały wybrane statystyki; odcinek, dla którego otrzymano odległość jednostkach rzeczywistych; wskaźnik, wskazujący na zmianę, posiadający etykietę tekstową	16
2.2 Przykład użycia aplikacji DWV - obrys półautomatyczny (Livewire)	17
2.3 Interfejs wgrywania plików DICOM do serwera Orthanc	18
2.4 Podgląd tagów pliku DICOM przy użyciu interfejsu przeglądarkowego Orthanc .	18
2.5 Podgląd obrazu DICOM przy użyciu interfejsu przeglądarkowego Orthanc	19
3.1 Diagram UML przedstawiający architekturę autorskiego systemu	25
3.2 Oryginalny obraz, który przedstawia badanie rezonansem magnetycznym stawu ramiennego	31
3.3 Obraz po zastosowaniu algorytmu usuwania niemaksymalnych krawędzi	31
3.4 Wynikowa macierz po zastosowaniu operatora Canny'ego, gdzie kolorem białym są reprezentowane wartości odpowiadające 0, a kolor czarny odpowiada 1	32
3.5 Ilustracja łączności 4-krotnej	33
3.6 Ilustracja łączności 8-krotnej	33
3.7 Przykładowe wygenerowane statystyki	41
4.1 Widok badania rezonansem magnetycznego barku	46
4.2 Widok zdjęcia rentgenowskiego barku	46
4.3 Widok badania rezonansem magnetycznego kręgosłupa	47
4.4 Widok badania tomografią komputerową kręgosłupa	47
4.5 Przykładowy obrys manualny	48
4.6 Przykładowy obrys półautomatyczny	48
4.7 Czasy generowania obrysu półautomatycznego kości ramiennej w zależności od ilości punktów	49
4.8 Czasy generowania obrysu półautomatycznego barku w zależności od ilości punktów	49

4.9	Czasy generowania obrysu półautomatycznego barku w zależności od ilości punktów drugą metodą	50
4.10	Jeden z wygenerowanych obrysów w trakcie testów	51
5.1	Widok listy pacjentów	72
5.2	Widok listy badań pacjenta	72
5.3	Widok listy serii w badaniu	73
5.4	Wybrany obraz w module obrysu manualnego	74
5.5	Przykładowy obrys	74
5.6	Okno zapisu obrysu manualnego	75
5.7	Podgląd wykonanego obrysu	76
5.8	Wybór punktu wewnętrz obrysu	77
5.9	Widok listy obrysów po zapisaniu obrysu manualnego	77
5.10	Moduł obrysu półauotomatycznego	78
5.11	Punkty wybrane do obrysu półauotomatycznego	79
5.12	Podgląd obrysu półauotomatycznego	79
5.13	Podgląd obrysu półauotomatycznego po dodaniu nowych punktów	80
5.14	Widok zapisu obrysu półauotomatycznego	81
5.15	Podgląd statystyk obrysu	81
5.16	Widok wyboru koloru obrysu	82
5.17	Widok wielu obrysów	82
5.18	Widok szczegółów obrazu	83
5.19	Anonimizacja danych pacjenta	83

Spis zawartości załączonej płyty CD

Do niniejszej pracy dyplomowej inżynierskiej została załączona płyta CD. W jej skład wchodzą:

- wersja elektroniczna pracy, której treść jest identyczna z powyższą w pliku **garsteckil-swierczewskit-praca-inzynierska.pdf**,
- strona tytułowa pracy w pliku **garsteckil-swierczewskit-strona-tytulowa.pdf**,
- streszczenie w języku polskim i angielskim w plikach o nazwach odpowiednio **garsteckil-swierczewskit-streszczenie.pdf** i **garsteckil-swierczewski-abstract.pdf**.

Kod źródłowy, który został podzielony na kilka katalogów:

- **DotNetProject/** - zawierający kod źródłowy głównego serwera aplikacji. Główne algorytmy odpowiedzialne za generowanie obrysów półautomatycznych znajdują się w podkatalogu **Logic/**,
- **Web/** - zawierający kod źródłowy aplikacji przeglądarkowej.

Spis załączników

1. Załącznik 1 - Pseudokod generujący graf z bitmapy

```
MATRIX - macierz wejściowa z oznaczonymi krawędziami jako 1
foreach(punkt A taki, że MATRIX(A) == 1)
{
    if ( punkt A ma 1 sąsiada albo co najmniej 3 sąsiadów )
    {
        // (tzn. jest albo punktem końcowym albo węzłowym)
        wstaw punkt A do kolejki wierzchołków L_V
        while ( kolejka wierzchołków L_V niepusta )
        {
            weź wierzchołek V_1 z kolejki L_V
            usuń V_1 z kolejki L_V
            dodaj wierzchołek V_1 do grafu G
            foreach (punkt B sąsiadujący z V_1 )
            {
                if ( MATRIX(B) == 1 )
                {
                    stwórz nową krawędź E.
                    dodaj B do E
                    ustaw wierzchołek V_1 jako początek krawędzi E
                    dodaj krawędź E do kolejki przetwarzanych krawędzi L_E
                }
            }
            while ( L_E niepusta)
            {
                weź krawędź E z L_E
                usuń E z kolejki
                stwórz kolejkę potencjalnych punktów krawędzi E, L_P
```

SPIS ZAŁĄCZNIKÓW

```
weź punkt C z E
usuń C z E
dodaj punkt C do kolejki L_P
while ( L_P niepusta )
{
    weź punkt D z L_P
    usuń D z L_P
    K = liczba sąsiadów D
    if ( K == 0 lub K > 1 )
    {
        stwórz wierzchołek V_2, który znajduje się w D
        do listy krawędzi wierzchołka V_2 dodaj E
        do listy krawędzi wierzchołka V_1 dodaj E
        ustaw wierzchołek V_2 jako koniec krawędzi E
        dodaj krawędź E do grafu G
        dodaj wierzchołek V_2 do kolejki L_V
    }
    if ( K == 1 )
    {
        dodaj punkt D do E
        foreach ( punkt F sąsiadujący z D )
        {
            if ( MATRIX(F) == 1 )
            {
                dodaj F do L_P
            }
        }
    }
    MATRIX(D) = 1
}
}
}
}
}
}
```

}

Zwróć graf G

2. Załącznik 2 - Instrukcja instalacji

W celu instalacji serwera Orthanc należy otworzyć stronę <https://www.orthanc-server.com/download.php>, pobrać wersję odpowiednią dla używanego systemu operacyjnego, a następnie postępować zgodnie z instrukcjami wyświetlonymi podczas instalacji. W celu uzyskania szczegółowych informacji odnośnie konfiguracji należy zapoznać się z dokumentacją znajdującej się na stronie <http://book.orthanc-server.com/users/cookbook.html>

W celu instalacji serwera obrysów i aplikacji webowej należy skopiować pliki znajdujące się na płycie w folderach Web oraz DotNetProject na stację, na której aplikacje te zostaną uruchomione.

Do działania serwera obrysów wymagany jest .NET Core w wersji 2.2 lub nowszej. W celu instalacji .NET Core należy otworzyć stronę <https://dotnet.microsoft.com/download>, pobrać wersję odpowiednią dla używanego systemu operacyjnego, a następnie postępować zgodnie z informacjami wyświetlonymi w trakcie instalacji.

W celu konfiguracji serwera w pliku DotNetProject\Api\Properties\launchSettings.json należy wpisać w 24 linii wartość adresu i portu na którym serwer ma zostać udostępniony, jako wartość pola applicationUrl . Przykładowa poprawna wartość linii 24 to `applicationUrl": "https://localhost:5001",`.

Ponadto w pliku DotNetProject\Api\Startup.cs należy podać w linii 33 link pod którym dostępna jest aplikacja przeglądarkowa jako argument metody WithOrigins. Przykładowa poprawna wartość linii 24 to `builder => builder.WithOrigins("http://localhost:8080", "http://localhost:3000")`.

W celu uruchomienia serwera obrysów należy przejść do skopowanego folderu DotNetProject/API i wykonać w konsoli polecenie `$ dotnet build && dotnet run`.

Do działania aplikacji przeglądarkowej wymagany jest Node w wersji co najmniej 9.4 oraz yarn w wersji co najmniej 1.10.1, aby zainstalować Node.js należy udać się na stronę <https://nodejs.org/en/download/> i wybrać wersję odpowiadającą systemowi, z którego korzystamy. Postępować zgodnie ze wskazówkami instalatora. W celu instalacji yarn należy udać się na stronę <https://yarnpkg.com/en/docs/install> aby pobrać wersję odpowiadającą systemowi, na którym będzie uruchamiana aplikacja i uruchomić instalator.

W celu konfiguracji portu, na który będzie wystawiona aplikacja webowa należy ustawić

SPIS ZAŁĄCZNIKÓW

odpowiednią liczbę w 4 linii pliku Web/express.js. Przykładowa poprawna konfiguracja:

```
$ const portNumber = 3000;
```

Ponadto w celu konfiguracji aplikacji należy ustawić adresy do API serwera obrysów i serwera Orthanc w pliku Web/src/helpers/requestHelper.ts poprzez zmianę zawartość cudzysłówów w pierwszych dwóch linijkach pliku. Przykładowa poprawna konfiguracja:

```
export const orthancURL = "http://localhost:8042/";  
export const apiURL = "https://localhost:5001/";
```

W celu uruchomienia aplikacji przeglądarkowej wchodzimy do folderu Web, w którym wykonujemy polecenie `$ yarn install && yarn prod`. Uwaga, tę komendę należy uruchomić w konsoli obsługującej skrypty w języku bash.

3. Załącznik 2 - Instrukcja użytkowania

Po otwarciu aplikacji użytkownik może przeglądać pacjentów (patients), których badania zostały wgrane do serwera Orthanc — Rysunek 5.1.

DICOM contour	
patients	
Anonymized1	∅
N/A	∅
Mass-Test_P_00016_LEFT_CC	∅
Mass-Test_P_00016_LEFT_MLO	∅
Mass-Test_P_00017_LEFT_CC	∅
Mass-Test_P_00017_LEFT_MLO	∅
Sample patient name	∅
Sample patient name	∅
name	∅
Tomka	∅

Rysunek 5.1: Widok listy pacjentów

Po wybraniu pacjenta poprzez kliknięcie lewym przyciskiem myszy na jego nazwę, na liście pojawiają się badania (studies) wybranego pacjenta. Użytkownik może wrócić do widoku pacjentów klikając w strzałkę na liście z badaniami — Rysunek 5.2.

studies	←
CMWUM^TRZUSTKA	└
CMWUM^TRZUSTKA	└

Rysunek 5.2: Widok listy badań pacjenta

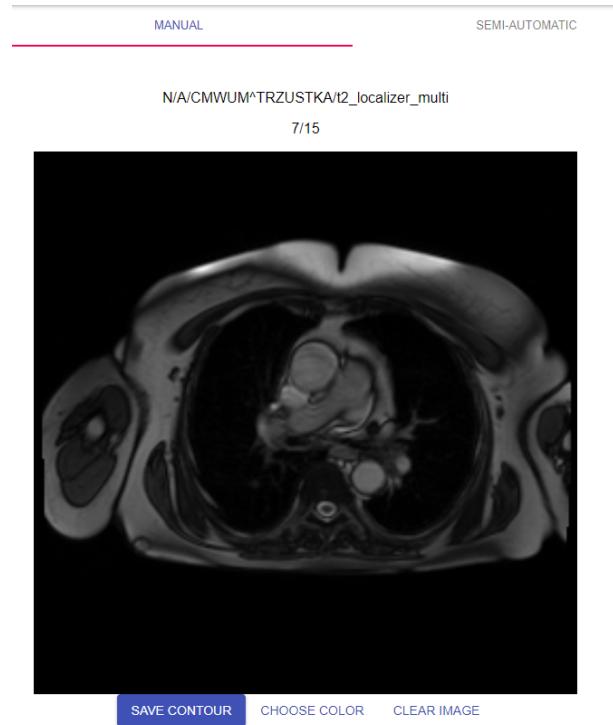
SPIS ZAŁĄCZNIKÓW

Po wybraniu badania poprzez kliknięcie lewym przyciskiem myszy na jego nazwę, na liście pojawiają się serie (series) wybranego badania. Użytkownik może wrócić do widoku badań, klikając w strzałkę na liście z seriami — Rysunek 5.3.

series	←
t2_localizer_multi	[img]
t2_blade_cor_mbh_fs	[img]
PosDisp: [2] t2_blade_cor_mbh_fs	[img]
t2_blade_tra_trigg_fs	[img]
PosDisp: [3] t2_blade_tra_trigg_fs	[img]
t2+t2_tse_tra_mbh_p2	[img]
ep2d_diff_tra_b0_50_100_150_200_400_800_1200_tr	[img]
ep2d_diff_tra_b0_50_100_150_200_400_800_1200_tr	[img]
t1_fl2d_in_opp_ph_tra_mbh	[img]
PosDisp: [9] t1_fl2d_in_opp_ph_tra_mbh	[img]
t2_haste_cor_thin_slab_mbh_3mm	[img]

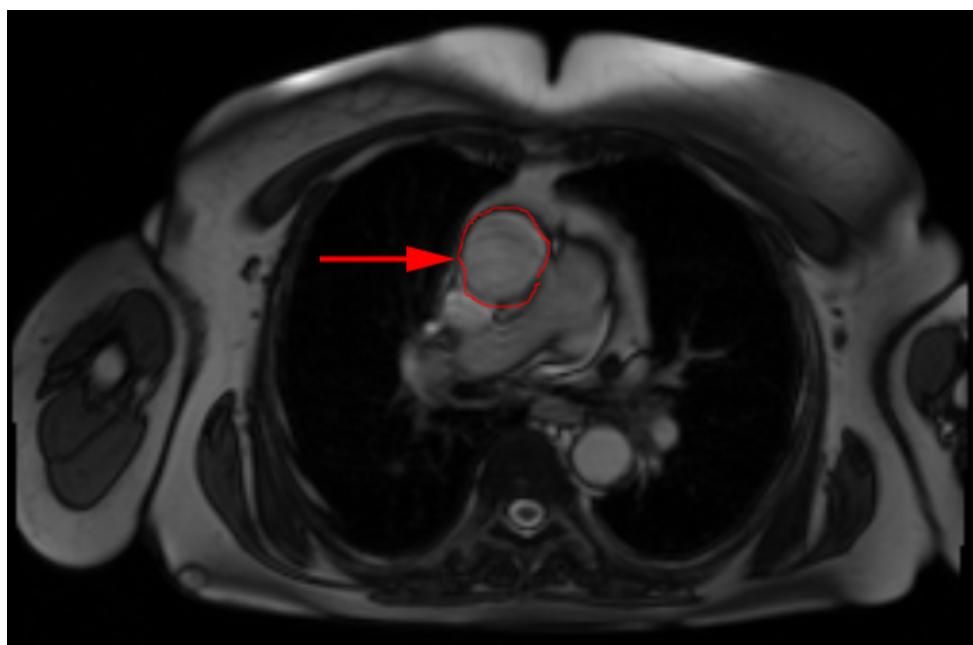
Rysunek 5.3: Widok listy serii w badaniu

Po wyborze serii na ekranie pojawia się pierwszy obraz z serii. Użytkownik może przełączać się pomiędzy obrazami korzystając z listy obrazów po lewej stronie lub używając rolki myszy po najechaniu na obraz. Aby powrócić do wyboru serii należy kliknąć lewym przyciskiem myszy na strzałkę na liście instancji (instances) — Rysunek 5.4.



Rysunek 5.4: Wybrany obraz w module obrysów manualnego

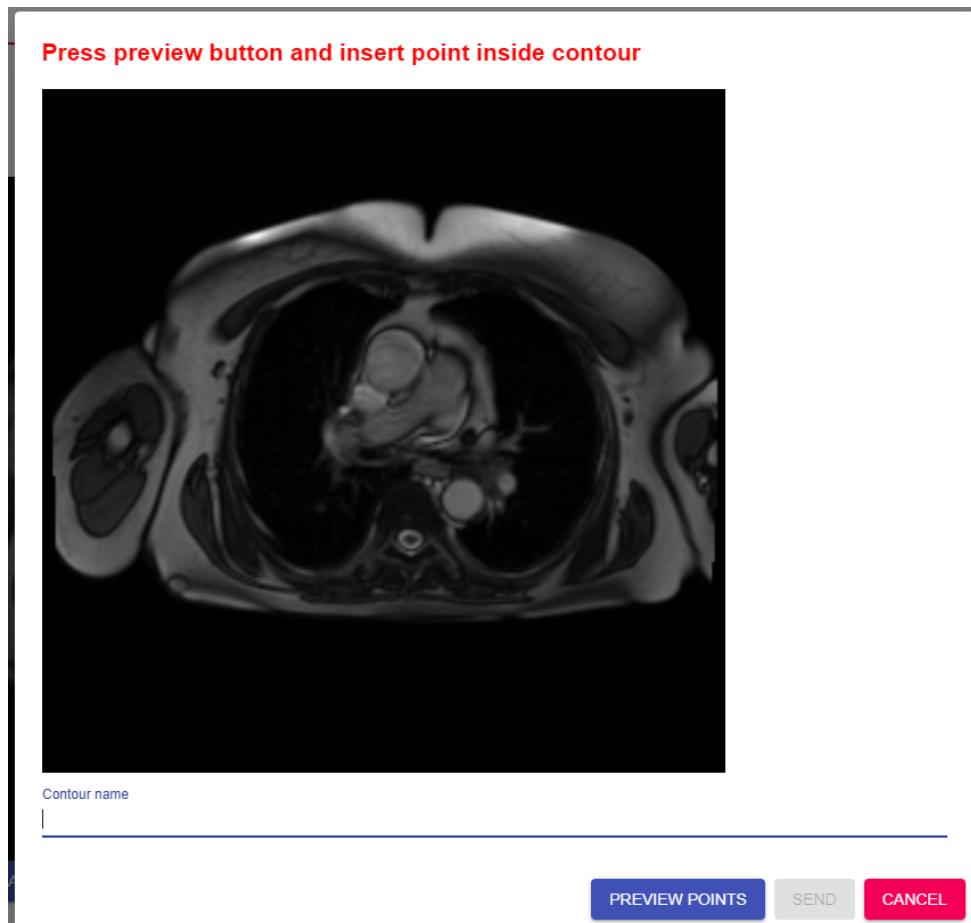
Przy manualnym obrysie użytkownik może wykonywać obrys poprzez przytrzymanie lewego przycisku myszy na obrazie, przesuwając mysz z wcisniętym przyciskiem. Gdy użytkownik korzysta z tabletu graficznego wystarczy, że będzie przesuwał wcisniętym rysikiem po tablecie — Rysunek 5.5.



Rysunek 5.5: Przykładowy obrys

SPIS ZAŁĄCZNIKÓW

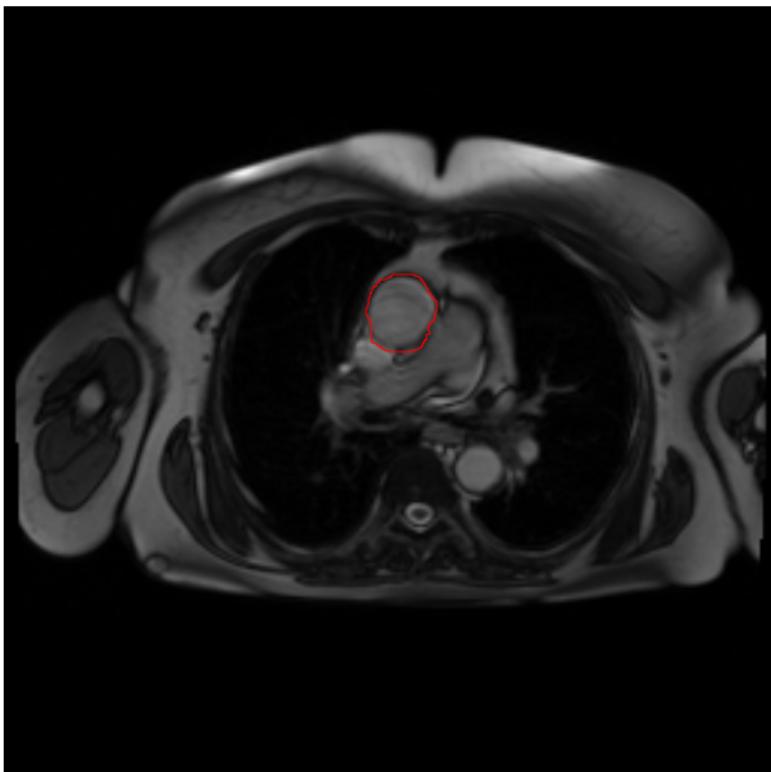
Po wciśnięciu przycisku zapisz obrys (Save Contour), użytkownikowi ukazuje się okno dialogowe. W celu usunięcia nieudanego obrysu należy kliknąć lewym przyciskiem myszy w przycisk wyczyść obraz (Clear Image). W celu zapisania obrysu należy wykonać następujące kroki: Wpisać nazwę obrysu w pole tekstowe poniżej obrazu (pod napisem Contour Name) — 5.6.



Rysunek 5.6: Okno zapisu obrysu manualnego

Po wpisaniu nazwy, należy kliknąć przycisk podgląd punktów (Preview Points) lewym przyciskiem myszy. Pozwoli to użytkownikowi zobaczyć wykonany przez niego obrys — Rysunek 5.7.

Press preview button and insert point inside contour

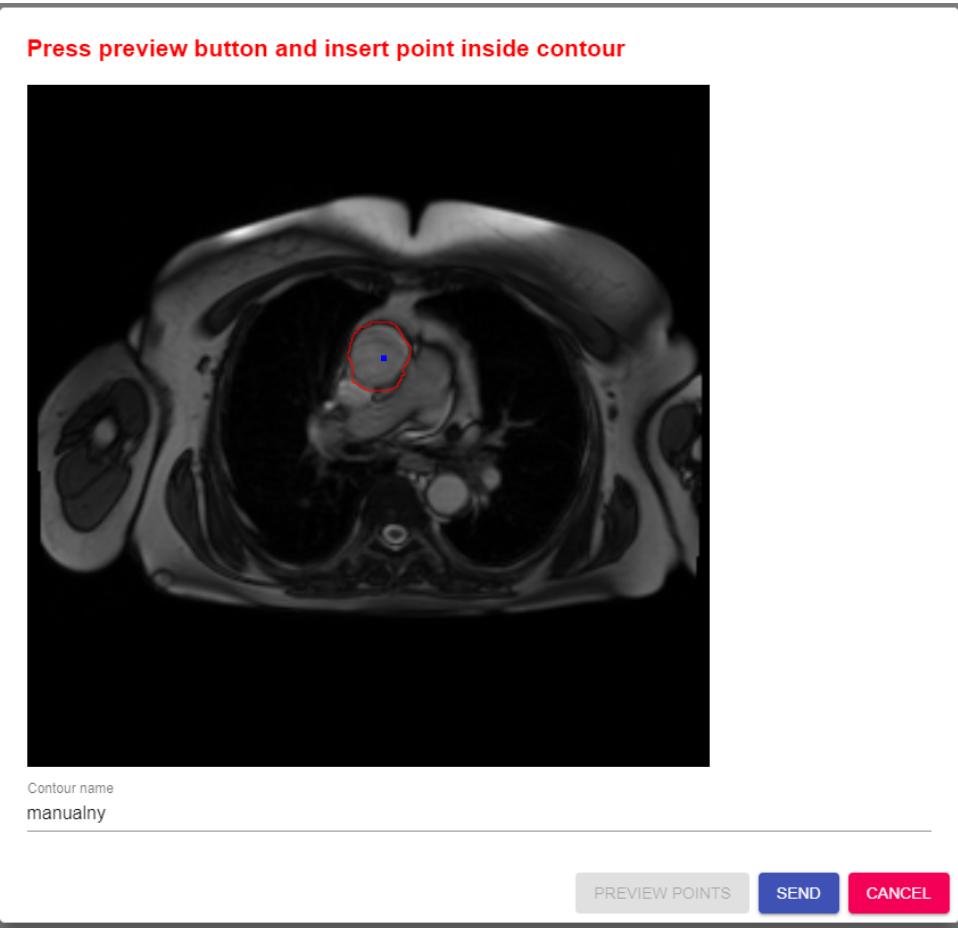


Contour name
manualny

PREVIEW POINTS **SEND** **CANCEL**

Rysunek 5.7: Podgląd wykonanego obrysu

Następną czynnością, którą musi wykonać użytkownik przed zapisaniem to wybranie punktu wewnętrz wykonanego obrysu, poprzez wcisnięcie lewego przycisku myszy w odpowiednim miejscu na obrazie — Rysunek 5.8.



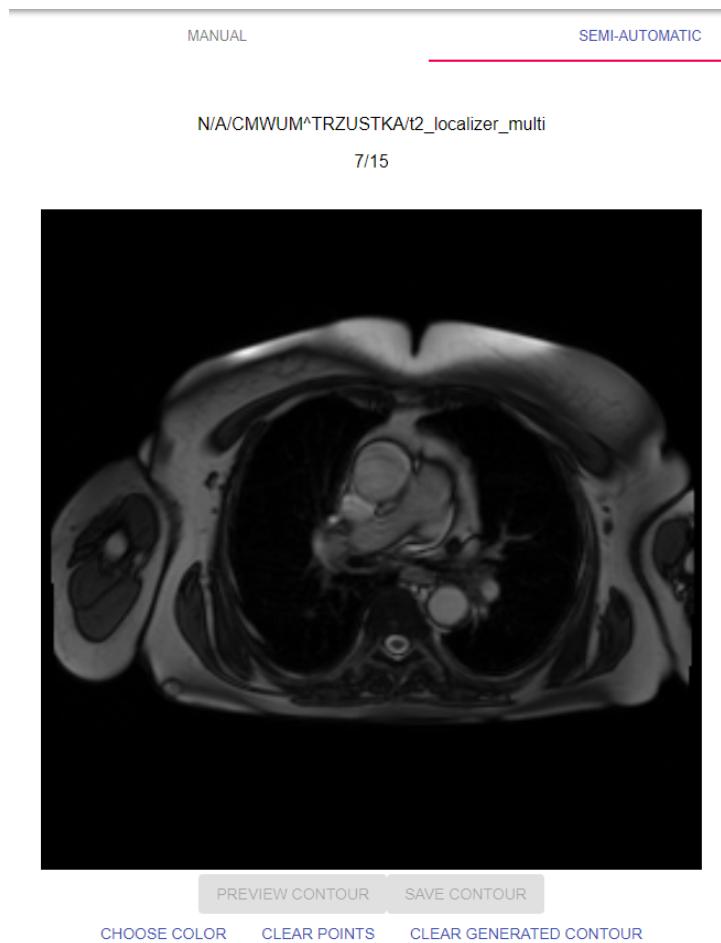
Rysunek 5.8: Wybór punktu wewnętrz obrysu

Po kliknięciu przycisku wyślij (Send) lewym przyciskiem myszy obrys zostaje zapisany i pojawia się na liście wykonanych obrysów (Available Contours). Lista zawiera jednie obrys dla oglądanego obrazu — Rysunek 5.9.

Available Contours	
nazwa 1	▷
manualny	▷

Rysunek 5.9: Widok listy obrysów po zapisaniu obrysu manualnego

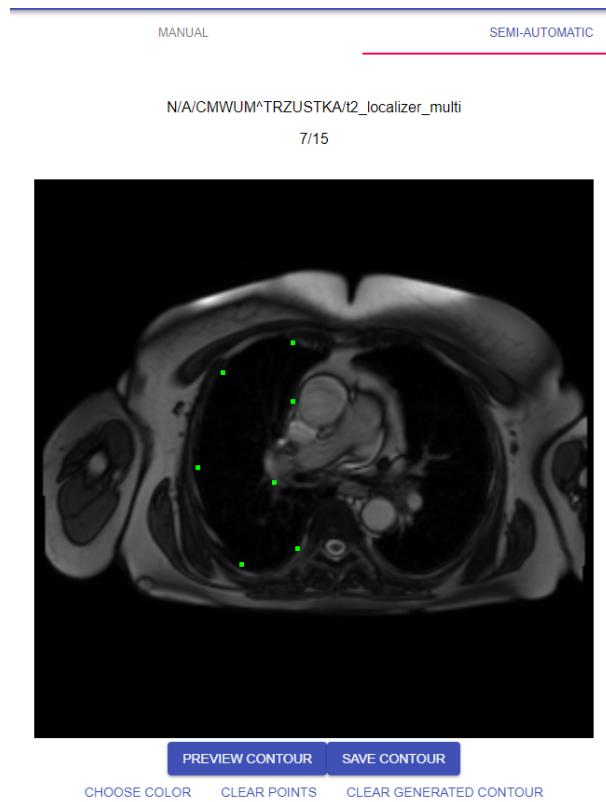
Użytkownik może wykonywać obrys w sposób półautomatyczny. W tym celu musi wybrać kartę z obrysami półautomatycznymi (z Manual do Semi-Automatic) — Rysunek 5.10.



Rysunek 5.10: Moduł obrysu półauotomatycznego

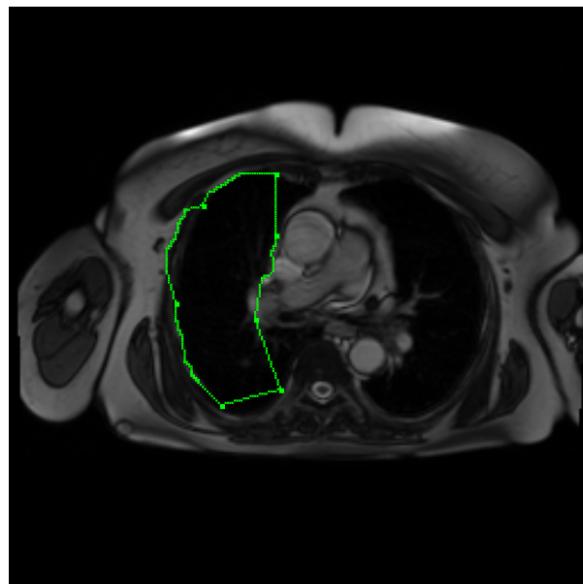
Aby wybrać punkty do obrysu półautomatycznego użytkownik powinien kliknąć w interesujące go punkty lewym przyciskiem myszy. Jeśli korzysta z tabletu graficznego, powinien przycisnąć rysik do tabletu w interesujących go punktach. Punkty wprowadzane powinny być w kolejności, w której występują w obrysie, zgodnie albo przeciwnie do ruchu wskazówek zegara — Rysunek 5.11.

SPIS ZAŁĄCZNIKÓW



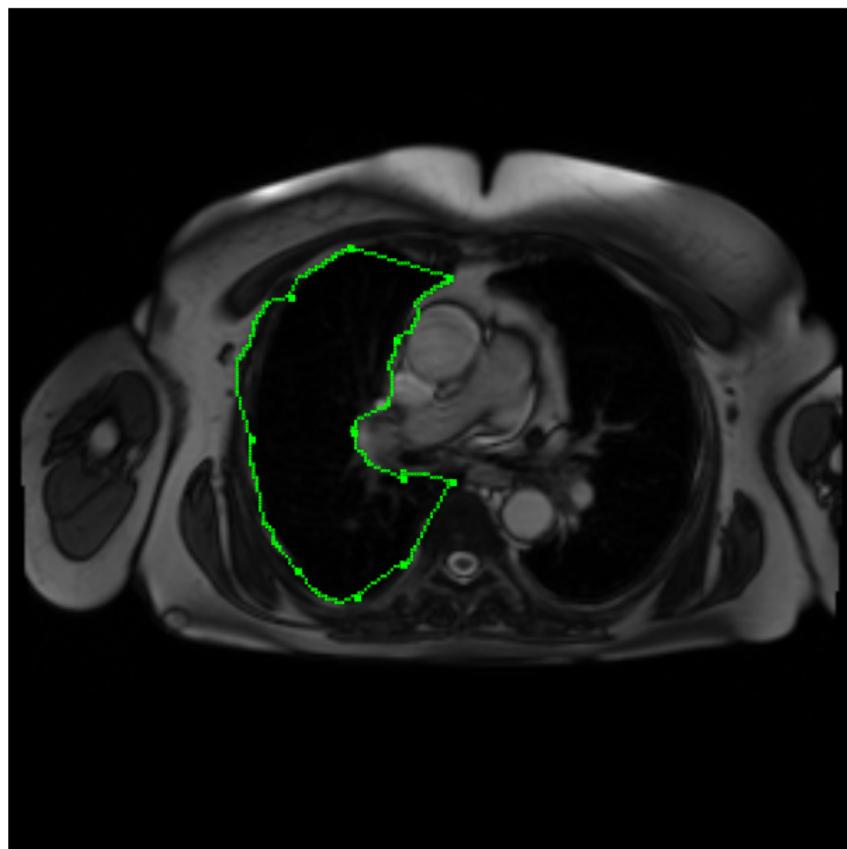
Rysunek 5.11: Punkty wybrane do obrysowania półauotomatycznego

Po wciśnięciu przycisku podgląd obrysu (Preview Contour) użytkownik może zobaczyć wygenerowany przez algorytm obrys. — Rysunek 5.12.



Rysunek 5.12: Podgląd obrysowania półauotomatycznego

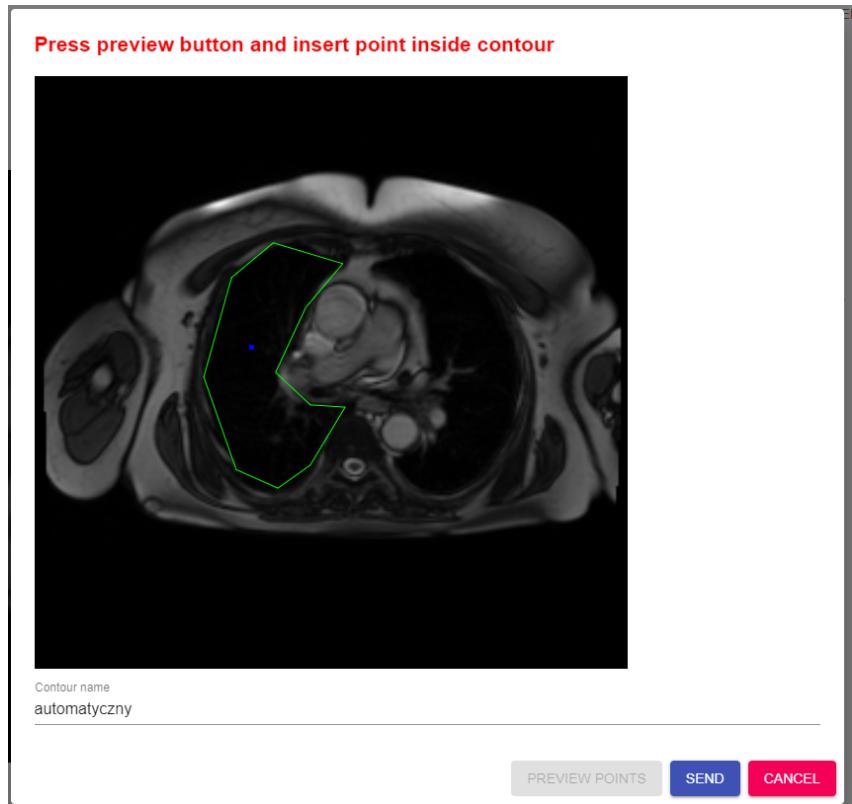
Jeśli algorytm popełnił błąd użytkownik może dodać nowe punkty klikając lewym przyciskiem myszy w interesujących go miejscach lub usunąć nietrafnie umieszczone punkty poprzez klikanie na nich lewego przycisku myszy — Rysunek 5.13.



Rysunek 5.13: Podgląd obrysu półautomatycznego po dodaniu nowych punktów

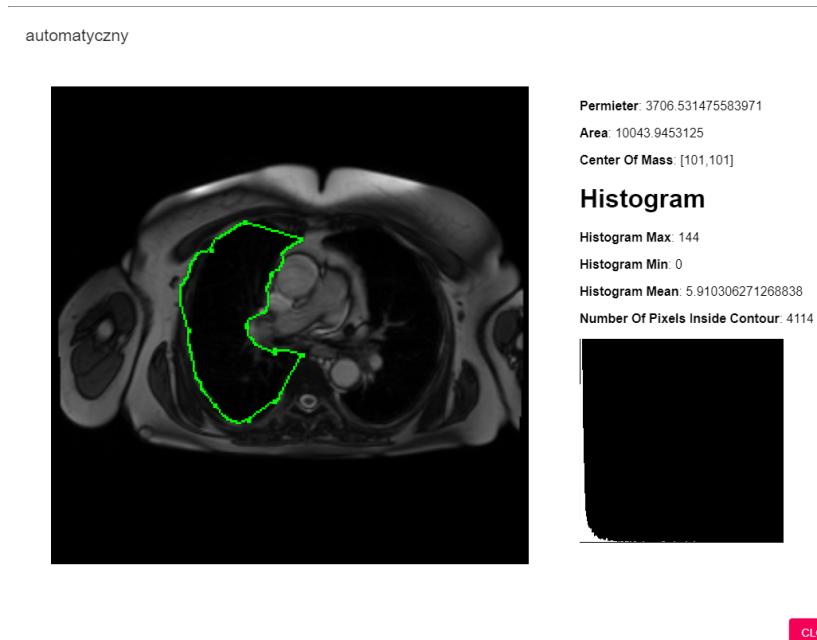
Gdy wygenerowany obrys jest satysfakcjonujący dla użytkownika, może go zapisać klikając zapisz obrys (Save Contour). Okno zapisu jest analogiczne do okna zapisu obrysu manualnego. Jeśli na obrazie pojawią się krzyżujące krawędzie, należy wyczyścić obrys i wykonać go od początku, gdyż algorytm modyfikujący obrys dodał nowe punkty w złym miejscu lub użytkownik wprowadził punkty w złej kolejności — Rysunek 5.14.

SPIS ZAŁĄCZNIKÓW



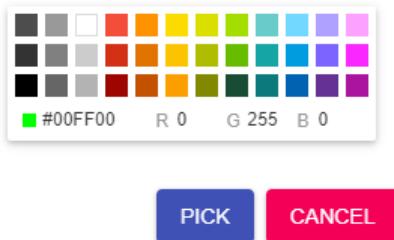
Rysunek 5.14: Widok zapisu obrysu półauotomatycznego

Po zapisaniu użytkownik może obejrzeć statystyki wykonanego obrysu poprzez kliknięcie lewego przycisku myszy na nazwie obrysu — Rysunek 5.15.



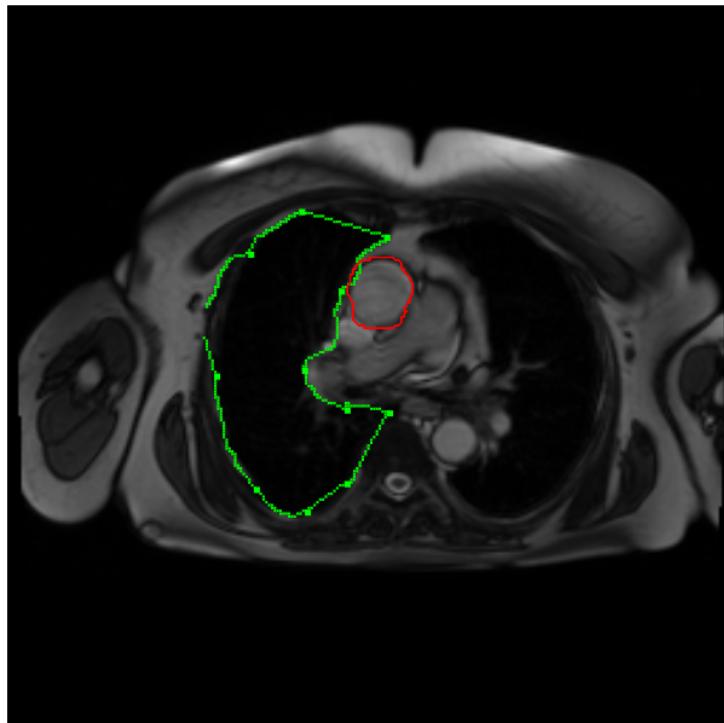
Rysunek 5.15: Podgląd statystyk obrysu

Przed wykonaniem obrysu użytkownik może wybrać kolor wykonywanego obrysu. Po wcisnięciu lewym przyciskiem myszy na przycisk wybierz kolor (Choose Color), pojawia się okno wyboru koloru. Użytkownik może wybrać kolor z przygotowanej wcześniej palety kolorów lub wprowadzić dowolny inny korzystając z pól tekstowych, wprowadzając kod RGB wybranego koloru — Rysunek 5.16.



Rysunek 5.16: Widok wyboru koloru obrysu

W zakładce z podglądem obrysów użytkownik może wyświetlać wybrane przez niego obrysy z listy wykonanych obrysów. Użytkownik wybiera obrysy, które chce zobaczyć. Obrysy, które nie są aktualnie wyświetlane można dodać do obrazu klikając je lewym przyciskiem myszy na nazwie obrysu, a wyświetlane można wyłączyć również poprzez kliknięcie lewego przycisku myszy na nazwie obrysu — Rysunek 5.17.



Rysunek 5.17: Widok wielu obrysów

SPIS ZAŁĄCZNIKÓW

Interfejs wyświetla również informacje dotyczące pacjenta i jego badania oraz wyświetlonego obrazu — Rysunek 5.18.

Name: N/A
Birthdate: 19340608
Sex: F
Institution Name: N/A
Referring Physician Name: N/A
Study Date: 20161010
Study Description: CMWUM^TRZUSTKA
Pixel Spacing: 1.5625\1.5625
Spacing Between Slices: 32
ANONYMIZE

Rysunek 5.18: Widok szczegółów obrazu

Poprzez wcisnięcie przycisku anonimizuj (Anonymize) użytkownik może wybrać nazwę pacjenta, datę jego urodzenia oraz płeć po anonimizacji jego danych — Rysunek 5.19.

Anonymize patient

Patient Name
N/A

Birthdate
19340608

Patient sex
F

ANONYMIZE **CANCEL**

Rysunek 5.19: Anonimizacja danych pacjenta